Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges

Giuseppe Aceto, Domenico Ciuonzo, *Senior Member, IEEE,* Antonio Montieri, and Antonio Pescapé, *Senior Member, IEEE*

***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. Traffic Classification (TC), i.e. the set of procedures for inferring (mobile) applications generating such traffic, has become nowadays the enabler for highly-valuable profiling information (with certain privacy downsides), other than being the workhorse for service differentiation/blocking. Nonetheless, the design of accurate classifiers is exacerbated by the raising adoption of encrypted protocols (such as TLS), hindering the suitability of (effective) deep packet inspection approaches. Also, the fast-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mobile traffic makes design solutions with usual machine learning, based on manually- and expert-originated features, outdated and unable to keep the pace. For these reasons Deep Learning (DL) is here proposed, for the first time, as a viable strategy to design practical mobile traffic classifiers based on automatically-extracted features, able to cope with encrypted traffic, and reflecting their complex traffic patterns. To this end, different state-of-the-art DL techniques from (standard) TC are here reproduced, dissected (highlighting critical choices), and set into a systematic framework for comparison, including also a performance evaluation workbench. The latter outcome, although declined in the mobile context, has the applicability appeal to the wider umbrella of encrypted TC tasks. Finally, the performance of these DL classifiers is critically investigated based on an exhaustive experimental validation (based on three mobile datasets of real human users' activity), highlighting the related pitfalls, design guidelines, and challenges.**

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

I. Introduction

VARIOUS tools, such as security/quality-of-service enforcement devices and network monitors, rely on the knowledge of the application generating the traffic and thus are limited (or impaired) when this requirement is not completely 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]. Notwithstanding, TC is challenged by the massive diffusion of handheld devices (as supported by recent evaluations in Internet usage [2]), which is revolutionizing the nature and the composition of traffic traversing home and enterprise networks and connecting contents and services over the Internet. Thus, the necessity and the difficulty of mobile TC have both become consistently high nowadays, fueled (other than common drivers for TC) by valuable profiling information (e.g. to advertisers, insurance companies, and security agencies) [3, 4], 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).

Equally important, the growing adoption [5, 6] of encrypted protocols (TLS) as well as network address translation and dynamic ports, poses new challenges to accurate TC, defeating established approaches such as deep packet inspection and port-based methods [7]. Indeed, the presence of Encrypted Traffic (ET) is a severe limitation that can be bypassed only in closed-world enterprise scenarios and adopting workarounds as man-in-the-middle proxies [8]. Moreover, other than the ET issue, mobile TC comes with exacerbated challenges and requirements due to (i) a large number of apps to discriminate from and (ii) the automatic frequent updates of the apps—leading to inadequate number of training samples per app and hindering the achievement of targeted performance.

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 [9, 10, 11], and they are also able to discriminate traffic generated from several apps.[1] However, the successful use of standard ML classifiers relies on obtaining handcrafted (domain-expert driven) features, which in TC context usually correspond to statistics extracted from the sequence of packets [9, 13] or message sizes [14, 15]. Sadly, 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 [10, 13, 16] with "traditional" ML approaches. Based on these considerations, we believe that Deep Learning (DL), which allows to train classifiers directly from input data by *automatically learning* structured (and complex) feature representations [17], may be the stepping stone toward high-performing TC in the dynamic and challenging mobile context.

Manuscript received 24th May 2018; revised 25th September 2018 and 31th January 2019.

Giuseppe Aceto and Antonio Pescapé are with the University of Napoli Federico II (Italy) and with NM2 s.r.l. (Italy). E-mail: {giuseppe.aceto,pescape}@unina.it.

Domenico Ciuonzo and Antonio Montieri are with the University of Napoli Federico II (Italy). E-mail: {domenico.ciuonzo,antonio.montieri}@unina.it.

[1] ML techniques can be also hybridized with port-association algorithms (in scenarios where port information can be considered reliable), e.g. [12].

However, a naïve adoption of DL techniques to (mobile) TC may imply *misleading* design choices and lead to *biased* conclusions, due to the peculiar (and tricky) nature of network traffic data. This constitutes, in our opinion, one of the main gaps to fill (viz. the prerequisite) for the capitalization of DL assets to mobile TC, thus echoing its successful use in "mature" fields, e.g. image and natural language processing [17].

Hence, this paper proposes for the first time (cf. Tab. II) the *design of mobile traffic classifiers (able to operate with ET) via the adoption of DL umbrella*. To this end, this work resorts on the development of a systematic framework for the design of novel DL-based TC architectures and comparison of existing ones, declined herein in the mobile scenario, but having a wider applicability to encrypted TC.[2] This originates from a critical analysis (later provided in Sec. II) of several non-mobile-specific DL classifiers recently appeared in TC literature [19, 20, 21, 22, 23, 24] and here reproduced (so as to avoid focusing on a specific DL technique and draw close-to-general conclusions).

In detail, the proposed framework dissects the DL-based TC problem from different viewpoints (highlighted via Fig. 1): (A) the TC object adopted, (B) the type (and the amount) of input data fed to the DL classifier, (C) the DL architecture employed, and (D) the required set of performance measures for an objective and comprehensive evaluation. Our framework is then applied to a realistic experimental setup, consisting of three different (mobile) datasets of *real human users' activity*, to assess the most appealing techniques, the potential gain w.r.t. ML-based best alternatives and shallow architectures (to justify the need for complex hierarchically-arranged features), and highlight open issues for real-time and accurate mobile TC via DL. Up to 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: (i) unbiased, informative, and heterogeneous inputs extrapolated from traffic data, (ii) sophisticated DL architectures, and (iii) a rigorous and multifaceted performance evaluation. This study represents a first attempt to address (i) and (ii) issues, being also a "safe" groundwork for paving the way to the design of accurate DL-based classifiers coping with highly-diverse mobile traffic, whereas it provides designers with a fine-level performance evaluation workbench (iii).

The rest of the paper is organized as follows. Sec. II reviews the related TC literature for the present work, whereas Sec. III describes the DL framework for mobile TC, focusing on key aspects to address, including the performance evaluation workbench here proposed; the experimental evaluation is reported and discussed in Sec. IV; finally, Sec. V provides lessons learned and highlights challenges. For the sake of readability, the acronyms used in this manuscript are summarized in Tab. I.

Table I: List of the acronyms used in the manuscript.

Acronym	Definition
AE / S(D)AE	AutoEncoder / Stacked (Denoising) AE
APDU	Application Protocol Data Unit
CNN	Convolutional Neural Network
CR	Classified Ratio
DL / ML	Deep/Machine Learning
ET	Encrypted Traffic
FB / FBM	FaceBook / FB Messenger
IAT	Inter-Arrival Times
LSTM	Long Short-Term Memory
MLP	MultiLayer Perceptron
PS	Packet Sizes
RF	Random Forest
RTPE	Run Time Per-Epoch
SVC	Support Vector Classifier
TC	Traffic Classification
WF	Website Fingerprinting

II. Background

Several recent works have dealt with TC of mobile apps, mainly under the assumption of ET. Nonetheless, TC in both a mobile and encrypted scenario by means of DL *appears currently unexplored*. To this end, we first describe foremost works performing TC in the mobile context, using standard ML-based approaches to deal with encrypted traffic. Then, we briefly discuss DL applied to the conceptually-similar task of Website Fingerprinting (WF), and, finally, we introduce the literature applying DL to Internet TC (i.e. not focusing on the mobile context). These three groups of related work are categorized in Tab. II, so as to sum up their main aspects and highlight their limitations in comparison to the present study.

A. Classification of Mobile Encrypted Traffic

A user fingerprinting scheme for devices that learns their traffic patterns by analyzing background activities (quantified as 70% of smartphone traffic) is developed by Stöber et al. [25]. Based on 3G transmissions, *bursts* of data are leveraged to extract statistical features which are used, via ML-based classifiers, to infer the specific user generating them. Differently, Wang et al. [26] propose a ML-based framework for app-usage classification considering Wi-Fi ET. Traffic frames are collected by running different iOS apps (among 8 categories) for 5 minutes. Results show an unexpected behavior of some apps with the increase of the training time, highlighting the lack of an accurate ground-truth labeling that affects classification performance. AppScanner is a framework for fingerprinting and identification of mobile apps developed by Taylor et al. [13]. An SVC and an RF are trained/tested with statistical (and raw) features extracted from the vector of the sizes of packets grouped based on timing and destination IP address/port aggregation criteria (*service burst*). App fingerprints are obtained by *automatically* running the 110 most popular apps from Google Play Store and preprocessing the traces to remove background traffic and TCP retransmissions and errors. Experimental results, other than satisfactory TC performance (shown in Tab. II), show an average 99% accuracy in single app identification. A comparison with state-of-the-art alternatives devised for the (conceptually-)similar WF task [31, 32] is also done, showing that AppScanner is able to significantly outperform them. More extensive analyses (on a larger dataset) of AppScanner are conducted in [10], to assess classification performance

[2]Preliminary results in the same framework of this study have been published as a conference publication [18].

Table II: Summary of previous works. First group adopted ML, second and third ones employed DL. Starred works aim at WF.

Paper	DL	ET	Mobile	Human	TC Object	Input Data	Classifier	Experimental Results
Stöber et al. [25]	○	●	●	●	Burst	Statistics of PS & IAT	SVC, K-Nearest Neighbors	$\geq 90\%$ acc. (20 users)
Wang et al. [26]	○	●	●	●	Burst	Statistics of PS & IAT	RF	$\approx 94\%$ acc. (13 iOS apps)
Taylor et. al. [10, 13]	○	●	●	○	Service burst	Statistics of PS	SVC, RF	86.9% acc. (110 Android apps)
Kampeas et al. [27]	○	●	○	○	Biflow	APDU sequence	C4.5 (J48)	$\geq 90\%$ acc. with 3 APDUs (54 classes)
Shahbar et al. [28]	○	●	○	●	Biflow	Statistics of PS & IAT	C4.5	$\approx 98\%$ acc. (22 classes)
Conti et al. [29]	○	●	○	○	TCP connection	Clustering-based features	RF	95% best acc. / prec. (up to 11 actions)
Alan and Kaur [30]	○	●	●	○	TCP connection	First 64 TCP PS	WF methods [31, 32]	88% best acc. (1595 Android apps)
Aceto et al. [16]	○	●	●	●	Service burst	Statistics of PS	Soft/Hard combination of traffic classifiers	$+9.5\%$ rec. w.r.t. best classifier (49 / 45 Android/iOS apps)
⋆ Oh et al. [24]	●	●	○	○	Tor cell sequence	Cell directions [784]	MLP, 2D-CNN, AE	92% / 1% rec. / fall-out (100 websites)
⋆ Rimmer et al. [33]	●	●	○	○	Tor cell sequence	Cell directions [150÷5k]	SDAE, CNN, LSTM	94% acc. (900 websites)
⋆ Sirinam et al. [34]	●	●	○	○	Tor cell sequence	Cell directions [5k]	SDAE, CNN	99% / 94% prec. / rec. (open-world)
Wang [19]	●	○	○	●	Biflow	TCP payload [1000 B]	SAE	$\geq 90\%$ prec. & rec. (25 protocols)
Zhang et al. [35]	●	●	○	●	Biflow	Manually-designed features	SAE	$\geq 90\%$ F-meas. (10 services)
Wang et al. [20]	●	●	○	●	Flow/Biflow	ALL/L7 layers [784 B]	2D-CNN	$\geq 89\%$ per-class metrics (up to 20 classes)
Wang et al. [21]	●	●	○	●	Flow/Biflow	ALL/L7 layers [784 B]	1D-CNN	$+2.51\%$ w.r.t. [20] (up to 12 classes)
Huang et al. [36]	●	●	○	●	Biflow	ALL layers [1024 B]	2D-CNN	$> 90\%$ per-class metrics (9 Trojans)
Chen et al. [37]	●	●	○	●	Biflow	L7-layer data and manually-designed features	Hierarchical DL with weighted backpropagation	99.6% / 85.4% acc. / prec. (12 classes)
Lotfollahi et al. [22]	●	●	○	●	Packet	IP packet [1500 B]	SAE, 1D-CNN	95% / 97% F-meas. (17 / 12 classes)
Lopez-Martin et al. [23]	●	●	○	●	Biflow	6 fields [20 packets]	Hybrid LSTM+2D-CNN	95.7% best F-meas. (108 services)
Shi et al. [38]	●	●	○	●	Biflow	ML&DL-selected features	Deep Belief Networks	$\approx 60\%$ G-mean (10 classes)
Vu et al. [39]	●	●	○	●	Biflow	Manually-designed features	Aux. Classifier Generative Adversarial Network	$\approx 95\%$ F-meas. (11 SSH & non-SSH services)
Li et al. [40]	●	○	●	●	HTTP session	HTTP fields [28×36 B]	Variational AE	99.6% acc. (12 Android apps)
This paper	●	●	●	●	*Biflow*	*ALL/L7 layers [256÷2304 B] 4 - 6 fields [4÷32 packets] Packet directions*	*SAE, LSTM, 1D-CNN, 2D-CNN, Hybrid LSTM+2D-CNN*	*Comprehensive evaluation (see. Sec. IV) E.g.* $\approx 86\%$ / $\approx 83\%$ *acc. (49 / 45 Android/iOS apps)*

degradation due to apps' fingerprint variation/aging because of different used device/app versions. Remarkable applications of decision trees are also found in [27, 28] for the encrypted TC problem, whereas the RF (with features obtained starting from a hierarchical clustering approach) is also employed in [29] for action fingerprinting of a certain app. The WF methods described in [31, 32] are also employed by Alan and Kaur [30] to check out whether Android apps can be identified from their launch-time traffic using only TCP/IP header information. In the best case considered, i.e. when training and test samples are collected on the same device, apps can be identified with 88% accuracy. Differently, a significant drop (up to 26% for the best classifier) is observed when the OS/vendor is different. Aging of training data caused by app updates is also taken into account. Recently, a novel multi-classification approach enjoying the fusion of state-of-the-art classifiers devised for mobile and encrypted TC is proposed in [16]. Four classes of combination techniques varying in accepted classifiers' outputs (i.e. soft or hard), training requirements, and learning philosophy are compared. Based on a dataset of *real users' activity* (as opposed to [10, 13, 29, 30], employing bot-generated mobile traffic), combination results present a performance gain according to all considered metrics.

B. Website Fingerprinting using Deep Learning

Henceforth, we first discuss the applications of DL to the (conceptually-similar) problem of (encrypted) WF. Oh et al. [24] study the usage of DL for WF and also prove its effectiveness on feature extraction (via AE) for state-of-the-art ML algorithms. Results underline that DL architectures successfully detect which website the user visited among 100 websites against 100k background websites. A novel DL-based method to deanonymize Tor traffic is proposed in [33] and tested on a very-large WF dataset made of $\geq 3 \cdot 10^6$ network traces. Results highlight that performance achieved via DL is comparable to state-of-the-art deanonymization attacks, with the best-performing DL model being $+2\%$ accurate. Finally, Sirinam et al. [34] develop a WF attack against Tor which is evaluated against state-of-the-art defenses (i.e. WTF-PAD and Walkie-Talkie). Performance evaluation in an open-world setting shows that the attack is effective against undefended traffic, while still relevant ($95/70\%$ of precision/recall) in case WTF-PAD defense is employed.

C. Standard Traffic Classification using Deep Learning

Herein we complete our review of related literature by discussing recent DL proposals to standard TC. Wang [19] suggests a first DL approach (based on SAE), applied to *clear* traffic identification (but adaptable to ET), and compares it to standard neural networks on a dataset made of 300k records stripped of duplication and HTTP traffic. Results show that SAE outperforms the latter and achieves high performance in protocol identification (taken from 58 different typologies) and a class prediction probability $\geq 80\%$ (resp. $\geq 90\%$) on 6.7k (resp. 5.5k) out of 10k traffic samples unrecognizable via deep packet inspection. The SAE (trained on *manually-designed features*) is also recently applied to TC in [35], showing that it outperforms a SVC and achieves a high F-measure on a

real-world traffic dataset. A novel malware TC, based on 2D Convolutional Neural Networks (CNNs) and explicitly devised for ET, is proposed in [20]. The approach is tested on a dataset ($\approx$ 752k instances) consisting of (i) 10 malware traffic types from public websites and (ii) 10 normal traffic types, in two different tasks: (i) malware vs. normal (binary) and (ii) traffic-type (20 classes) classification. Also, two different choices of raw "traffic images" (named "ALL" and "L7") dependent on the protocol layers considered to extract the input data, are used to feed the classifier, showing that (biflow-based) TC with "ALL" is the most informative and reaches elevate performance for all the metrics considered. In [21] the same authors propose a similar approach for encrypted TC based on a 1D-CNN. Experiments, conducted on a selection of the "ISCX VPN-nonVPN" dataset [41], consist of four different 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). Consistently with [20], the configuration "Biflow + ALL" performs the best and the configuration-optimized 1D-CNN always achieves higher accuracy than a 2D-CNN counterpart (being both however $\geq$ 80%) in all the setups, and (almost) always outperforms the C4.5 classifier originally designed by Gil et al. [41].

More recently, in [36] the problem of handling multiple TC problems (i.e. malware detection, recognition of VPN-encapsulation, and Trojan classification) at once via a single 2D-CNN DL architecture is tackled. The 2D-CNN is tested on a merge of "CTU-13" (malware) and "ISCX VPN-nonVPN" traffic datasets, and shown to outperform each element of comparison for each task considered. Another study on DL-based TC of malware is represented by [37], dealing with issues of an imbalanced dataset via "weighted" backpropagation and a hierarchical approach exploiting *both* raw data and handcrafted features. Experimental results on a self-generated dataset show that the proposed approach outperforms standard ML/DL alternatives. The same dataset is also used to test *Deep Packet* [22], a DL-based (namely 1D-CNN and SAE) framework for encrypted TC working at packet-level. Deep Packet is compared to state-of-the-art ML classifiers on the same dataset, showing to outperform the latter in both application identification (K-Nearest Neighbors) and traffic characterization (C4.5).

Different DL architectures for encrypted TC, based on hybrid compositions of Long Short-Term Memory (LSTM) and 2D-CNN layers, are proposed in [23]. The best-performing of these variants (named "CNN+RNN-2A" therein) attains $\geq$ 95% metrics on a dataset captured on Spanish academic backbone network 'RedIRIS' and made of 266k biflows from 108 distinct services. The analysis also highlights (i) a performance drop by including inter-arrival times in the input and (ii) that $5 \div 15$ packets are enough for satisfying results. In [38] a novel feature optimization approach, based on deep belief networks and ML-based feature selection techniques is devised to improve TC performance, by overcoming the negative impacts of multi-class imbalance and concept drift. Experiments on real traffic show that the approach outperforms existing ML classifiers and a deep belief network without feature selection. Another application of DL to TC with imbalanced network data is found in [39], where an auxiliary-classifier generative adversarial network is used to generate synthesized samples (in the form of a set of *handcrafted* features) for training set balancing, to be used by ML classifiers. The method, tested on NIMS dataset, outperforms a counterpart based on the synthetic minority over-sampling technique.

To the best of our knowledge, the sole application of DL to mobile TC, other than our preliminary work [18], seems to be [40], where a DL classifier, based on variational autoencoders and input data taken from the reconstructed HTTP session (i.e. designed only for *clear traffic*) is proposed and tested on a self-generated dataset.

III. Framework for comparison and tuning of Deep Learning–based Traffic Classification

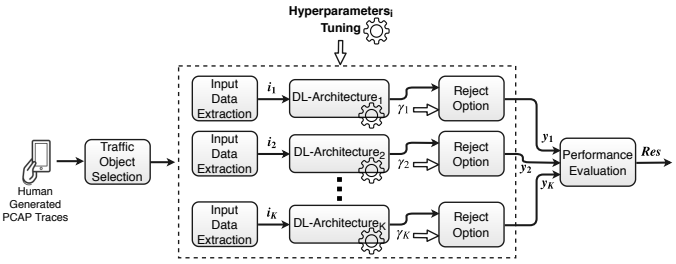


Figure 1: Framework for comparison and tuning of DL architectures for TC.

This section dissects the state-of-the-art of DL in TC, by focusing on the following *viewpoints*: (A) the traffic object (i.e. the type of traffic aggregate, also known as traffic view [1]), (B) the types of input data extracted to feed the DL architectures, (C) the DL architectures employed, and (D) the performance evaluation measures used. Based on these points, Fig. 1 sketches out the framework devised for the systematic comparison of DL-based traffic classifiers. It is worth pointing out that all the DL classifiers proposed for TC have been carefully analyzed and reproduced, e.g. by setting the hyperparameter 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 [42] (Python) API running on top of TensorFlow to implement and test the approaches described in the following.

A. Traffic Object

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 [19, 20, 21, 23] considered either *flows* or *biflows* as the relevant objects of classification, with the sole exception of [22]. 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 [22] the object of classification is the *single packet* (i.e. the classification

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 [19, 20, 21];
II the first N bytes of raw data pertaining to the PCAP file related to the TC object [20, 21];
III informative data fields of first N_p packets [23].

Based on the aforementioned categorization, it is worth noticing that all the types of input data considered for DL are naturally suited for "early" TC [43].

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 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 [19] these correspond to the first 1000 bytes of TCP payload. A similar choice is made in [20, 21] for the input labeled as "L7", where 784 bytes from the *application layer* in TCP/IP model are considered. Differently, in [22] 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.[3]

The *second* type of input data attempts to gather information from *all protocol layers* (denoted with "ALL" layers in [20, 21]) 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 [20, 21]. 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.[4] Specifically, in [20, 21] 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 [23] 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[5], inter-arrival time, and packet direction ($\in \{0, 1\}$). We highlight that the (binary-valued) sequence of packets/messages directions has been also recently employed in DL-based WF [24, 33].

[3]Additionally, the authors apply also a pre-processing step to cope with unequal transport-layer header lengths, by padding with zeros the end of the UDP-datagram headers up to TCP-segment headers length.

[4]We underline that extraction of "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 16-byte header, including a timestamp at μs granularity and packet size information.

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

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. Deep Learning–based Classification Architectures

Herein we review the architectures employed for DL-based TC. For convenience, we define the m^{th} instance of the training set (made of M samples) as $\boldsymbol{x}_{(m)}$ while the corresponding label with $\ell_{(m)}$, belonging to one among L different classes (i.e. $\ell_{(m)} \in \{1, \ldots, L\}$). All the considered DL classifiers are trained to minimize the categorical cross-entropy [17]:

$$\mathcal{L}(\cdot) \triangleq \sum_{m=1}^{M} \left\{ -\sum_{l=1}^{L} t_{l,(m)} \, \log p_{l,(m)} \right\} \tag{1}$$

In the above equation, the one-hot representation of the label $\boldsymbol{p}_{(m)} \triangleq \begin{bmatrix} p_{1,(m)} & \cdots & p_{L,(m)} \end{bmatrix}^T$ and of the corresponding predicted vector $\boldsymbol{t}_{(m)} \triangleq \begin{bmatrix} t_{1,(m)} & \cdots & t_{L,(m)} \end{bmatrix}^T$ are employed. The minimization of the loss $\mathcal{L}(\cdot)$ is achieved by means of standard (first-order) local optimizers (e.g. stochastic gradient descent, adaptive moment estimation, etc.), resorting to the usual back-propagation for gradients evaluation.

SAE: The SAE (Fig. 2(a)) relies on the basic AutoEncoder (AE), commonly employed for (unsupervised) feature learning, and whose aim is to (ideally) set the output $\boldsymbol{y}_{(m)} \approx \boldsymbol{x}_{(m)}$, $\forall m = 1, \ldots, M$, by learning a *compressed* data representation. Specifically, the first AE block (i.e. the *encoder*) provides a lower-dimensional data representation (via a hidden layer of neurons), whereas the second block (i.e. the *decoder*) tries to reconstruct the data from the compressed representation.

In practice, to obtain improved performance, a more complex (hierarchical) architecture, namely the SAE, has been proposed [17]. This scheme employs *unsupervised greedy layer-wise pre-training* (top part of Fig. 2 (a)) 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 (i.e. each layer of network is trained by keeping the weights of lower layers frozen). After training greedily AE layers, a final softmax layer is added and supervised *fine-tuning* (i.e. a refinement of all layers' weights) of the whole network (bottom part of Fig. 2 (a)) for the classification task is performed (i.e. using $\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(M)}$ along with $\ell_{(1)}, \ldots, \ell_{(M)}$). A relevant application of SAE to TC is found in [22], consisting of *five* stacked layers—with $\{400, 300, 200, 100, 50\}$ neurons and 25% dropout-probability [17] after each layer (to mitigate over-fitting)—all employing rectified linear unit activations.

CNN: The CNNs (Fig. 2 (b)) are widely-used DL models, inspired by visual mechanism of living organisms, and made of chained convolutional layers, each comprising a set of translation-invariant *filters* (conceived in either 1D or 2D form, depending on the specific input nature) with a limited extent (the "receptive field") which are *convolved* with the input with the aim of extracting features of a certain input region. Another important CNN component is the *pooling*

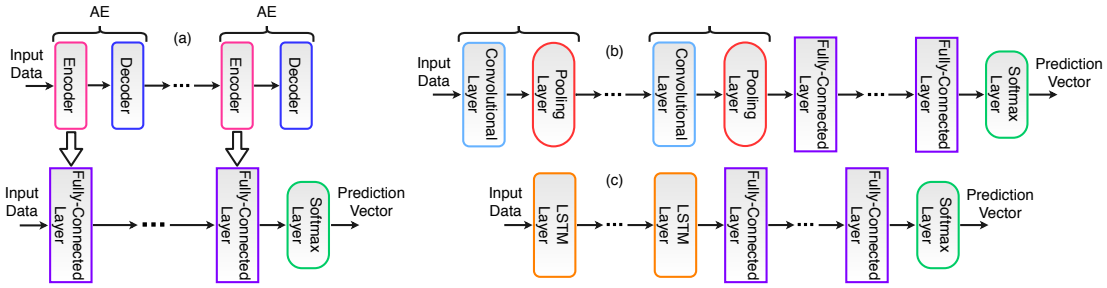


Figure 2: DL architectures for TC: SAE (a), CNN (b), LSTM (c).

layer, typically following a convolutional layer 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 fully-connected (similar to AE compressing stage) layers, with the last having the essential softmax activation. For example, the architecture in [21] 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.

Similarly, the CNN in [20] is obtained by replacing 1D with 2D (pooling/convolutional) layers and interpreting the input as a "traffic image". A similar 2D-CNN is also considered in [23], where *batch normalization* [17] is also applied after each max-pooling layer. Differently, in [22] a 1D-CNN consisting of *two* 1D convolutional layers (200 and 80 filters, respectively, with 1D average-pooling) and *seven* fully-connected layers (with $\{600, 500, 400, 300, 200, 100, 50\}$ neurons), all having rectified linear unit activations, is considered. Additionally, to avoid the over-fitting, 25% dropout after pooling layer and *early stopping* technique are adopted [17].

LSTM: An LSTM (Fig. 2 (c)) is a popular (easier to train) variant of recurrent neural networks (having unit connections forming a directed cycle), able to model *dynamic* temporal behaviors with "long-term dependencies" [17]. A neural network made of LSTM units is often called an LSTM network.

An LSTM unit is in charge of "remembering" values (via a state vector $\boldsymbol{h}[t]$) over arbitrary time intervals and is composed of a *cell*, *input*, *output*, and *forget* gates, while having as input a vector sequence of length T: $\boldsymbol{x}[1], \ldots, \boldsymbol{x}[T]$ (i.e. each training instance is a matrix). The final hidden state $\boldsymbol{h}[T]$ corresponds to the output of LSTM unit. A standard 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, they expose as output (except for the last one) the finer-grained time-evolution of the state vs. the input sequence, $\boldsymbol{h}[1], \ldots, \boldsymbol{h}[T]$ (modeling a "return-sequences" behavior), forming the input to the higher LSTM layer.[6] For example in [23] a standard 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 [23] in the context of hybrid architectures, as described henceforth.

[6]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.

Hybrid DL Architectures: The discussed elementary learning layers can be also jointly employed within a single DL architecture. For example, architectures based on the combination of 2D convolutional and LSTM layers may be conceived [23], where the output tensor of the convolutional layer is reshaped into a matrix fed as input to an LSTM unit.

D. Performance Evaluation Workbench

The proposed comparison framework includes the following common performance measures [1]: accuracy (the fraction of correctly classified instances), precision (prec, i.e. the proportion of classifier decisions for a given class which are actually correct), recall (rec, i.e. the class-conditional accuracy), and specificity (spec, i.e. the proportion of actual negatives of a class that are correctly identified as such). Since the latter three are defined on a per-app basis, we consider the *F-measure* $\mathrm{F} \triangleq (2 \cdot \mathrm{prec} \cdot \mathrm{rec})/(\mathrm{prec} + \mathrm{rec})$ and the *G-mean* $\mathrm{G} \triangleq \sqrt{\mathrm{rec} \cdot \mathrm{spec}}$ so as to account for their effects concisely, and employ their arithmetically averaged (viz. macro) versions. Moreover, the concept of *Top-K accuracy* (recently used in WF [24]) is employed, defining a correct classification event if the true app is within the top K predicted labels ($K < L$ is a free parameter[7]) and allowing to investigate the soft-output of a DL classifier. Finally, we consider also *confusion matrices* 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 [10]. Indeed, since apps typically send multiple flows where used, there remains high chance to identify them from their more distinctive flows, without the need to classify all the instances (i.e. the classifier does not reach a verdict when the highest class prediction probability is below γ). Hence, tuning γ can be effective to improve classification performance while incurring negligible drawback, i.e. a decreased ratio of classified instances (CR).

For completeness, as a preliminary investigation of the computational complexity of DL-architectures training phase, we report their training runtime, given the specificity of such phase in mobile TC, due to apps' fingeprint aging because of their (and OSs) updates. Precisely, since training is performed on multiple epochs [17], we report such info in a terse (normalized) way, by providing the Run-Time Per-Epoch (RTPE).

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

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

IV. Experimental Evaluation

The present section investigates and compares performance of considered DL classifiers, according to Sec. III-D, 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* (instead of relying on automatically-generated traffic, as done in related works). 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 [22]), we have chosen to operate at the *biflow level* when referring to the traffic view.

Multi-class datasets: The first two (multi-class) datasets, obtained from a global mobile solutions provider and generated from 49 (resp. 45) apps on Android (resp. iOS) devices, are considered for prioritization purposes.[8] The corresponding Android (resp. iOS) traces have been collected during Apr. '15 - Jan. '17 (resp. Sept. '14 - Jan. '17), generated by users with different devices and OS/app versions, and provided already anonymized and cleaned from background traffic. In detail, $\approx 89\%$ (resp. $\approx 85\%$) of Android (resp. iOS) traces has been captured in '16. As a whole, the dataset is made up of 607 (resp. 419) traffic traces, with mean duration of 282 (resp. 296) seconds and $1 \div 60$ (resp. $1 \div 48$) traces per app in Android (resp. iOS), leading to a non-negligible class imbalance. Such realistic setup justifies the need for a complete evaluation framework of DL-based classifiers, as proposed in Sec. III. Finally, after *biflow* segmentation, 77.3k (resp. 44.1k) labeled instances compose the Android (resp. iOS) dataset, with 73.8k (resp. 41.8k) TCP and 3.5k (resp. 2.3k) UDP biflows.

FB/FBM binary dataset: The third (binary) dataset has been collected in ARCLAB laboratory at the University of Napoli "Federico II", during several sessions within May '17 - Mar. '18 timespan. More specifically, the captures pertain to either Facebook (FB) or Facebook Messenger (FBM) traffic data, and run on a Xiaomi Mi5 with Android Operating System 6.0.1 (CyanogenMod 13.0 distribution). This choice derives from the peculiar nature of these two apps, both devoted to interactive usage of the Facebook platform (author of both). This suggests a high possibility of shared development framework and overlapping services usage, hampering the discrimination of the respective traffic (as suggested by the same provider and also confirmed experimentally in next section) needed for key management tasks e.g. for billing differentiation. More than 100 users have been involved in its construction on a voluntary basis for sittings lasting less than 2 hours, being required to perform different activities for both the apps (to explore their diversity), in union with login/registration/logged-in use cases. Each traffic-capture session lasted $5 \div 10$ minutes, with > 1100 traffic traces collected. As a whole, the dataset contains > 34k instances, with 15.0k (resp. 19.2k) biflows generated by FBM (resp. FB) app, with a $44\%/56\%$ share. Precisely, FBM (resp. FB) traffic consists of 13.2k (resp. 18.7k) TCP and 1.8k (resp. 0.5k) UDP biflows, respectively.[9]

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 (both multi-class and binary), varying the actual number of biflows.

B. Baselines Considered and Classification Results

We now provide a systematic comparison of the considered DL architectures so as to draw out key guidelines (later elaborated in Sec. V). For completeness, *two baseline approaches* are also included in our analysis of classification efficacy: (i) the RF developed in [13], i.e. the current state-of-the-art mobile-traffic classifier, taking as input 40 carefully handcrafted *flow-based features* and thus applicable only in "post-mortem" TC (as opposed to inputs used in DL classifiers, suited for "early" TC), and (ii) a MLP with only one hidden layer (with 100 nodes), denoted as MLP-1, trained on the same inputs as DL architectures, so as to stress the performance achievable by shallow learning in the same scenario.

Hereinafter, we refer to Type I (resp. Type II, cf. Sec. III-B) input data corresponding to the first N bytes of payload (resp. raw) data as "L7-N" (resp. "ALL-N") [19, 20, 21]. Differently, the 20×4 (resp. 20×6, when ports are included, highlighted through a "$\star$" marker) input matrix obtained following [23] (Type III) is denoted with "MAT", with the general notation "MAT-N_p" when a varying number of packets is considered. Finally, for consistency, the first N_p packet directions (Type III) are referred to as "DIR-N_p" [24].

Biased vs Unbiased Inputs: First, in Tabs. III and IV 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, respectively. We highlight that performance of classifiers marked with diamond markers ($\diamond$) represents results from *biased inputs* (cf. Sec. III-B) and, therefore, they *should not* be considered as *meaningful elements of comparison*. From the inspection of results it is apparent that, referring to multi-class datasets (cf. Tab. III), DL approaches are able to provide improved performance w.r.t. 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. This is attributed to DL ability to learn implicitly very complex features able to distinguish (seemingly) similar traffic generated from different apps. Indeed, in Android setup, 85.46% accuracy, 78.78% F-measure, and 86.92% G-mean are achieved by 2D-CNN (L7-784), as opposed to 84.78%, 75.49%, and 83.86%, respec-

[8] 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.

[9] The current dataset constitutes a larger version w.r.t. that considered in [18], in terms of both depth and diversity, while improving also the balance between FB/FBM samples (i.e. $44\%/56\%$ vs. $38\%/62\%$ share of [18]).

Table III: Accuracy, F-measure, and G-mean [%] 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. Results with diamond (◇) and star (⋆) markers refer to *biased* inputs and inputs *including TCP/UDP ports*, respectively. **Best-performing** DL-based and shallow classifiers fed with *unbiased* inputs are highlighted for both datasets.

Architecture	Android			iOS		
	Accuracy	F-measure	G-mean	Accuracy	F-measure	G-mean
SAE [22] (L7-1000)	75.15 (± 1.52)	57.00 (± 2.78)	69.07 (± 3.42)	74.55 (± 0.80)	60.57 (± 2.06)	74.86 (± 1.89)
2D-CNN [20] (L7-784)	85.46 (± 0.48)	**78.78 (± 1.39)**	**86.92 (± 1.26)**	**82.72 (± 1.47)**	**74.41 (± 0.90)**	83.91 (± 0.95)
2D-CNN [20] (ALL-784) ◇	95.74 (± 0.24)	92.05 (± 0.65)	95.15 (± 0.56)	95.27 (± 1.19)	92.48 (± 0.91)	95.41 (± 0.76)
1D-CNN [21] (L7-784)	**85.70 (± 0.45)**	78.68 (± 1.20)	86.82 (± 0.87)	82.64 (± 1.63)	74.34 (± 1.29)	**84.00 (± 1.31)**
1D-CNN [21] (ALL-784) ◇	95.73 (± 0.67)	92.18 (± 1.19)	95.42 (± 1.02)	95.97 (± 0.38)	92.33 (± 0.99)	95.45 (± 0.67)
2D-CNN [23] (MAT) ⋆	82.22 (± 0.42)	70.81 (± 0.97)	82.18 (± 0.79)	81.23 (± 0.73)	73.04 (± 1.33)	83.64 (± 1.03)
LSTM [23] (MAT) ⋆	81.18 (± 0.41)	69.68 (± 0.81)	81.21 (± 0.65)	83.54 (± 0.64)	75.95 (± 1.11)	85.88 (± 0.89)
LSTM + 2D-CNN [23] (MAT) ⋆	83.53 (± 0.41)	72.02 (± 0.77)	82.51 (± 1.01)	82.28 (± 0.42)	74.22 (± 0.93)	84.36 (± 0.92)
2D-CNN [23] (MAT)	76.01 (± 0.70)	62.83 (± 1.28)	75.60 (± 1.29)	68.53 (± 0.61)	58.67 (± 1.22)	72.95 (± 1.30)
LSTM [23] (MAT)	73.64 (± 1.56)	59.53 (± 1.40)	73.31 (± 1.01)	66.50 (± 1.03)	56.27 (± 1.73)	71.98 (± 1.45)
LSTM + 2D-CNN [23] (MAT)	77.95 (± 0.41)	64.52 (± 1.17)	76.35 (± 1.45)	69.17 (± 0.64)	58.75 (± 0.76)	72.17 (± 0.75)
2D-CNN [24] (DIR-784)	40.11 (± 0.56)	15.41 (± 0.82)	24.61 (± 1.18)	32.95 (± 0.65)	11.42 (± 0.62)	18.18 (± 1.06)
MLP-2 [24] (DIR-784)	27.94 (± 0.82)	4.51 (± 0.22)	8.94 (± 0.26)	21.17 (± 0.44)	4.15 (± 0.32)	8.00 (± 0.59)
MLP-1 (L7-1000)	77.76 (± 0.38)	67.85 (± 1.45)	79.75 (± 1.29)	76.11 (± 0.84)	66.95 (± 1.47)	79.63 (± 1.44)
MLP-1 (L7-784)	**78.71 (± 0.65)**	**69.79 (± 1.17)**	**81.52 (± 1.38)**	**77.16 (± 0.63)**	**67.61 (± 1.07)**	**80.11 (± 0.99)**
MLP-1 (ALL-784) ◇	96.53 (± 0.27)	94.28 (± 0.72)	96.80 (± 0.54)	97.24 (± 0.50)	95.29 (± 0.81)	97.15 (± 0.65)
MLP-1 (MAT) ⋆	72.54 (± 0.47)	58.29 (± 1.11)	71.87 (± 1.27)	66.94 (± 0.90)	56.51 (± 1.24)	70.88 (± 1.08)
MLP-1 (MAT)	64.94 (± 0.47)	48.26 (± 0.96)	63.10 (± 1.07)	54.42 (± 0.63)	40.86 (± 1.04)	57.56 (± 1.03)
RF [13] (flow-based)	84.78 (± 0.30)	75.49 (± 0.89)	83.86 (± 0.58)	80.77 (± 0.84)	72.39 (± 1.39)	81.88 (± 1.27)

Table IV: Accuracy, F-measure, and G-mean [%] of DL-based and baseline traffic classifiers. Results refer to FB/FBM dataset and are in the format *avg. (± std.)* obtained over 10-folds. Results with diamonds (◇) and stars (⋆) refer to *biased* inputs and inputs *including TCP/UDP ports*, respectively. **Best-performing** DL-based and shallow classifiers fed with *unbiased* inputs are highlighted.

Architecture	Accuracy	F-measure	G-mean
SAE [22] (L7-1000)	73.52 (± 0.82)	71.82 (± 1.31)	70.49 (± 2.25)
2D-CNN [20] (L7-784)	75.56 (± 3.15)	73.95 (± 2.54)	71.81 (± 2.07)
2D-CNN [20] (ALL-784) ◇	73.99 (± 3.03)	72.54 (± 2.80)	70.85 (± 3.33)
1D-CNN [21] (L7-784)	**76.37 (± 0.73)**	**75.56 (± 1.01)**	**74.79 (± 1.76)**
1D-CNN [21] (ALL-784) ◇	75.91 (± 2.74)	75.53 (± 2.68)	75.46 (± 2.61)
2D-CNN [23] (MAT) ⋆	71.82 (± 1.13)	70.84 (± 1.12)	70.01 (± 1.07)
LSTM [23] (MAT) ⋆	72.59 (± 0.75)	71.76 (± 0.78)	71.10 (± 0.85)
LSTM + 2D-CNN [23] (MAT) ⋆	72.36 (± 0.95)	71.41 (± 0.96)	70.58 (± 1.04)
2D-CNN [23] (MAT)	73.33 (± 0.93)	72.18 (± 1.04)	71.02 (± 1.16)
LSTM [23] (MAT)	73.54 (± 0.49)	72.50 (± 0.58)	71.49 (± 0.85)
LSTM + 2D-CNN [23] (MAT)	73.74 (± 0.69)	72.66 (± 0.72)	71.58 (± 0.82)
2D-CNN [24] (DIR-784)	66.51 (± 0.57)	63.88 (± 0.82)	61.28 (± 1.23)
MLP-2 [24] (DIR-784)	58.93 (± 0.80)	56.65 (± 2.20)	54.73 (± 3.83)
MLP-1 (L7-1000)	73.78 (± 1.09)	72.58 (± 1.16)	71.95 (± 1.43)
MLP-1 (L7-784)	**74.46 (± 0.88)**	**73.89 (± 0.86)**	**73.55 (± 0.89)**
MLP-1 (ALL-784) ◇	76.39 (± 0.96)	75.82 (± 0.90)	75.42 (± 0.91)
MLP-1 (MAT) ⋆	68.66 (± 0.99)	67.65 (± 1.13)	66.88 (± 1.45)
MLP-1 (MAT)	68.93 (± 1.32)	67.86 (± 0.94)	66.98 (± 0.75)
RF [13] (biflow-based)	79.56 (± 0.62)	78.73 (± 0.62)	78.37 (± 0.76)

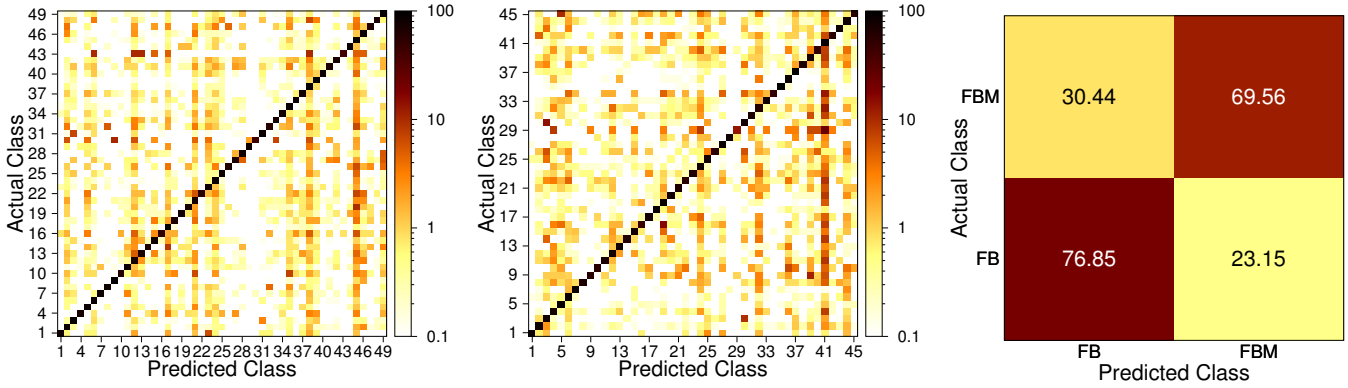
tively, obtained by the RF. We notice that, in both datasets, 1D-CNN (L7-784) achieves very similar performance to 2D-CNN (L7-784). This result confirms the intuition that discriminative information from *traffic should be extracted by naturally considering data as one-dimensional* (viz. time-series). A similar reasoning applies to iOS case, where LSTM performs the best in terms of the three considered metrics, but only when *port information is taken into account* (i.e. with "MAT ⋆" input). Differently, a significant performance drop is observed for each DL classifier with "MAT" input compared to its counterpart including both source and destination TCP/UDP ports in the input ("⋆" marker). For example, up to -19.68% in F-measure is observed for multi-class datasets, with the worst drop affecting LSTM in the iOS case. Finally, referring to the FB/FBM dataset (cf. Tab. IV), only the 2D-CNN (L7-784) is able to outperform the shallow classifiers MLP-1 (L7-1000/L7-784) in terms of all the metrics analyzed. Nonetheless, in the binary dataset neither the best DL classifier is able to achieve performance comparable with biflow-based RF. This may be attributed to the need of a more informative type of input, providing a higher discriminative power in the case of very similar apps, like FB and FBM. Finally, focusing on the DL approaches with "MAT" input, results highlight a *different trend* w.r.t. the multi-class datasets, with FB/FBM classification task almost being *port-independent*, showing even a slight performance gain (e.g. $+1.51\%$ accuracy with a 2D-CNN (MAT)) when ports are removed. This may be the consequence of high port randomization or/and (likely) use of overlapping port sets (e.g. corresponding to common services).

Top-K Accuracy: Delving into performance of DL-based classifiers, in Tab. V we report their Top-K accuracy ($K \in \{1, 3, 5\}$) on the multi-class datasets. From now on we exclude, for brevity, the results of DL classifiers based on biased inputs. By looking at these fine-grained results, we observe that, other than the highest DL accuracy, 1D-CNN (resp. 2D-CNN) (L7-784) reports also the highest global (soft-output) behavior on the Android (resp. iOS) dataset, e.g. 91.51% and 93.45% (resp. 91.02% and 93.32%) accuracy when the Top-3 and Top-5 predicted apps are considered, respectively.[10] Also, although shallow (baseline) classifiers present an accuracy increase due to a larger pool of predicted apps taken into consideration, they are never able to approach the same score as the best DL classifiers, confirming also an improved global behavior of the latter (viz. learning of the TC task as a whole).

[10]Still, 1D-CNN (L7-784) performs almost on par on iOS dataset.

Table V: Top-K accuracy [%] 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. Only the classifiers fed with *unbiased* inputs are shown. **Best-performing** DL-based and shallow classifiers are highlighted for both datasets.

Architecture	Android			iOS		
	$K=1$	$K=3$	$K=5$	$K=1$	$K=3$	$K=5$
SAE [22] (L7-1000)	75.15 (± 1.52)	82.16 (± 0.85)	85.53 (± 0.72)	74.55 (± 0.80)	82.73 (± 0.92)	86.58 (± 0.79)
2D-CNN [20] (L7-784)	85.46 (± 0.48)	91.36 (± 0.31)	93.35 (± 0.30)	**82.72 (± 1.47)**	**91.02 (± 0.42)**	**93.32 (± 0.33)**
1D-CNN [21] (L7-784)	**85.70 (± 0.45)**	**91.51 (± 0.27)**	**93.45 (± 0.29)**	82.64 (± 1.63)	90.95 (± 0.36)	93.29 (± 0.32)
2D-CNN [23] (MAT)	76.01 (± 0.70)	86.49 (± 0.53)	90.32 (± 0.39)	68.53 (± 0.61)	82.75 (± 0.46)	87.96 (± 0.36)
LSTM [23] (MAT)	73.64 (± 1.56)	85.58 (± 0.58)	89.93 (± 0.50)	66.50 (± 1.03)	81.94 (± 0.88)	87.23 (± 0.73)
LSTM + 2D-CNN [23] (MAT)	77.95 (± 0.41)	87.38 (± 0.37)	90.80 (± 0.29)	69.17 (± 0.64)	82.23 (± 0.38)	87.16 (± 0.39)
2D-CNN [24] (DIR-784)	40.11 (± 0.56)	58.88 (± 0.56)	68.29 (± 0.52)	32.95 (± 0.65)	53.91 (± 0.72)	64.40 (± 0.63)
MLP-2 [24] (DIR-784)	27.94 (± 0.82)	42.02 (± 0.26)	51.75 (± 0.27)	21.17 (± 0.44)	40.40 (± 0.55)	50.84 (± 0.64)
MLP-1 (L7-1000)	77.76 (± 0.38)	85.96 (± 0.30)	89.11 (± 0.20)	76.11 (± 0.84)	85.86 (± 0.65)	89.48 (± 0.51)
MLP-1 (L7-784)	**78.71 (± 0.65)**	**86.93 (± 0.40)**	**89.88 (± 0.37)**	**77.16 (± 0.63)**	**86.96 (± 0.50)**	**90.40 (± 0.51)**
MLP-1 (MAT)	69.94 (± 0.47)	79.22 (± 0.51)	84.94 (± 0.34)	54.42 (± 0.63)	72.47 (± 0.59)	80.03 (± 0.56)
RF [13] (flow-based)	84.78 (± 0.30)	91.69 (± 0.31)	93.89 (± 0.24)	80.78 (± 0.79)	90.70 (± 0.61)	93.58 (± 0.52)



(a) 1D-CNN [21] (L7-784). (b) 2D-CNN [20] (L7-784). (c) 1D-CNN [21] (L7-784).

1 = 360Security, 2 = 6Rooms, 3 = 80sMovie, 4 = 9YinZhenJing
5 = Anghami, 6 = BaiDu, 7 = Crackle, 8 = EFood
9 = FrostWire, 10 = FSecureVPN, 11 = Go90, 12 = Google+
13 = GoogleAllo, 14 = GoogleCast, 15 = GoogleMaps, 16 = GooglePhotos
17 = GooglePlay, 18 = GroupMe, 19 = Guvera, 20 = Hangouts
21 = HidemanVPN, 22 = Hidemyass, 23 = Hooq, 24 = HotSpot
25 = IFengNews, 26 = InterVoip, 27 = LRR, 28 = MeinO2
29 = Minecraft, 30 = Mobily, 31 = Narutom, 32 = NetTalk
33 = NileFM, 34 = Palringo, 35 = PaltalkScene, 36 = PrivateTunnelVPN
37 = PureVPN, 38 = QQ, 39 = QQReader, 40 = QianXunYingShi
41 = RaidCall, 42 = Repubblica, 43 = RiyadBank
44 = Ryanair, 45 = SayHi, 46 = Shadowsocks
47 = SmartVoip, 48 = Sogou, 49 = eBay

(d) Android Labels

1 = 360Security, 2 = 6Rooms, 3 = 80sMovie, 4 Anghami
5 = AppleiCloud, 6 = BaiDu, 7 = Brightcove, 8 = Crackle
9 = EFood, 10 = FSecureVPN, 11 = Go90, 12 = Google+
13 = GoogleAllo, 14 = GoogleCast, 15 = GoogleMaps, 16 = GooglePhotos
17 = GroupMe, 18 = Guvera, 19 = Hangouts, 20 = HiTalk
21 = HidemanVPN, 22 = Hidemyass, 23 = Hooq, 24 = HotSpot
25 = IFengNews, 26 = LRR, 27 = MeinO2, 28 = Minecraft
29 = Mobily, 30 = Narutom, 31 = NetTalk, 32 = NileFM
33 = Palringo, 34 = PaltalkScene, 35 = PrivateTunnelVPN, 36 = PureVPN
37 = QQReader, 38 = QianXunYingShi, 39 = Repubblica
40 = Ryanair, 41 = SayHi, 42 = Shadowsocks
43 = Sogou, 44 = eBay, 45 = iMessage

(e) iOS Labels

Figure 3: Confusion matrices of the best DL-based classifier for the (a) Android, (b) iOS, and (c) FB/FBM datasets. Note that the log scale is used to evidence small errors (except for FB/FBM). Categorical class-labels are reported for the (d) Android and (e) iOS datasets.

Such "global" performance gap is even more apparent for DL classifiers resorting to packet directions, whose best Top-5 accuracy is only 68.29% (resp. 64.40%) in Android (resp. iOS) case. Hence, although mobile TC can be conceived as a conceptually-similar task to WF, it shows higher requirements w.r.t. the former, since the sole directions are usually sufficient for training of high-performing WF classifiers [24, 33]. Finally, the (flow-based) RF classifier provides a slightly better global behavior than the best DL classifier on the Android dataset, reaching 91.69% (resp. 93.89%) Top-3 (resp. Top-5) accuracy.

Confusion Matrices: Turning to the details of classifiers behavior, Fig. 3 shows the confusion matrices of best-performing DL-approaches in the three datasets, so as to investigate noteworthy error-patterns.[11] From inspection of the results, the 1D-CNN (L7-784) (in Android and FB/FBM datasets) and 2D-CNN (L7-784) (in iOS dataset) achieve almost-uniform error patterns. The FB/FBM matrix contrasts, only at a first look, the earlier result shown in [18], referring to an older (smaller and class-imbalanced) version of the dataset. However, the results on the current (balanced) dataset are not significantly better, implying that the main error source on FB/FBM arises from the *inadequacy* of the considered pairs of input and DL architecture, as well as the *traffic similarity* of the two apps.

[11] Since 1D-CNN (L7-784) and 2D-CNN (L7-784) perform about on par on the multi-class dataset, we have chosen the one with the highest accuracy.

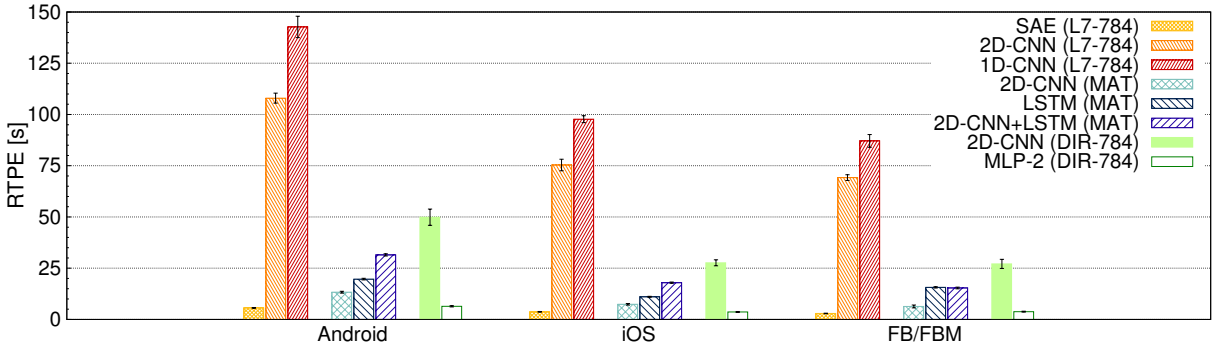


Figure 4: Run-Time Per Epoch (RTPE) of DL-based traffic classifiers. Results are in the format *avg. (± std.)* obtained over 10-folds. Only the classifiers fed with *unbiased* inputs are shown.

Training Complexity of DL Architectures: To investigate the training complexity of the considered DL classifiers, in Fig. 4 we report their RTPE obtained in the three datasets.[12] Results highlight a natural RTPE decrease of each classifier when the size of the classification problem is reduced (i.e. moving from the Android dataset, to the iOS and FB/FBM datasets). Additionally, the two classifiers reaching the highest performance are those having the highest RTPE (i.e. 2D-CNN (L7-784) and 1D-CNN (L7-784)), highlighting a reasonable performance-complexity tradeoff. Referring to the aforementioned two classifiers, we remark that 1D-CNN (L7-784) experiences a higher RTPE than 2D-CNN (L7-784) because of lower size of the pooling layers (i.e. lower down-sampling) in its implementation [20, 21]. On the other hand, all DL classifiers based on "MAT" input present a significantly lower complexity, being this a direct consequence of the lower-dimension input set ($20 \times 6 = 120$ as opposed to 784). Analogous considerations apply to DL classifiers based on "DIR-784" input, having a lower complexity than those based on "L7-784", because the former are binary valued, with the 2D-CNN (DIR-784) having a higher complexity w.r.t. MLP-2 (DIR-784), because of its more complex architecture. Finally we highlight that Fig. 4 reports, for the SAE, only the RTPE score corresponding to the *fine-tuning* phase (i.e. in which the SAE is trained in a supervised fashion as a "deep" MLP) and thus neglects its *pre-training stage*, which contributes additively to RTPE with a linear growth in the number of AE layers (since it is done in a layer-wise fashion).[13]

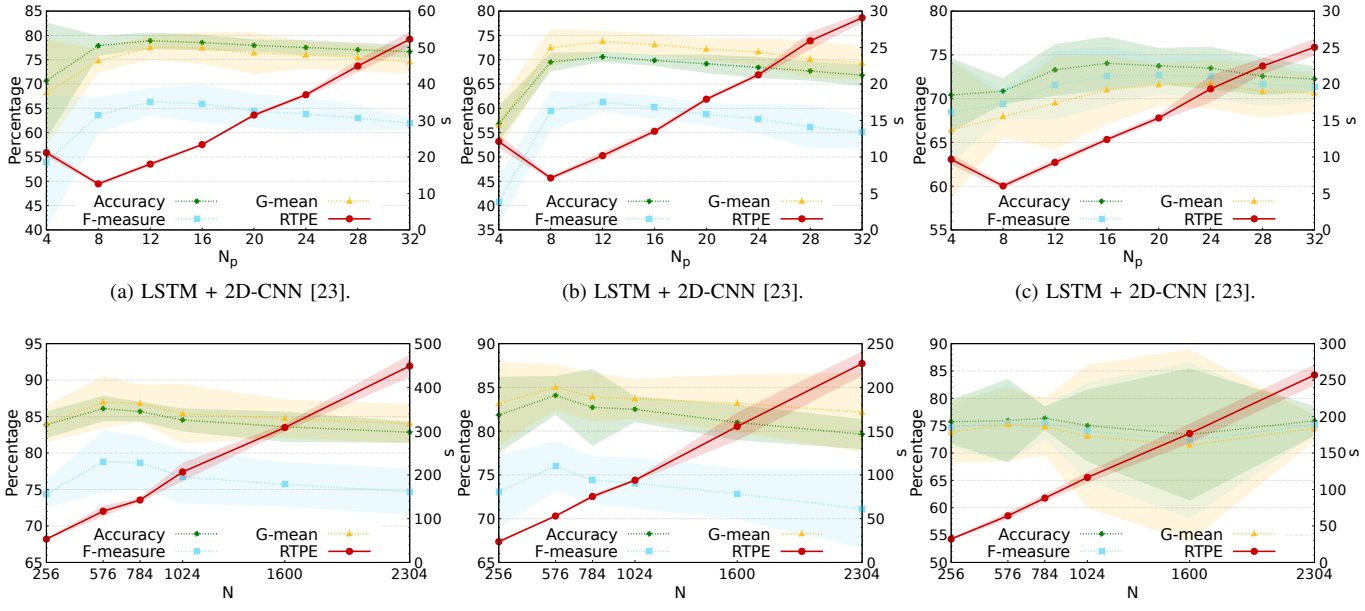
Performance vs. Input Size: Focusing our investigation toward the choice of the most discriminative forms of input types, in Fig. 5 we report accuracy, F-measure, and G-mean for the best DL classifier based on two types[14] of (unbiased) input data considered herein (i.e. "MAT-N_p" and "L7-N") vs. the number of packets N_p and payload bytes N, respectively. To highlight the relevant input size-complexity trade-off, we also report the RTPE measure vs. the size of the considered input data. From the inspection of results, it is apparent that, in the case of N_p input (Fig. 5 (a-c)), there is a *unimodal behavior* and $16-20$ packets are usually enough to achieve the highest performance (denoting a higher requirement w.r.t. the results shown in [23]), whereas, in the payload size case (Fig. 5 (d-f)), such *trend is less obvious* (although $N = 784$ is observed to be the best choice among the different sizes considered). On the other hand, in both cases an *almost-linear* increase of the RTPE with the input size is apparent. The only exception is given by $N_p = 4$: the reason is that, so as to implement the same DL architecture with a very small input, we had to resort to a different padding choice, implying additional complexity.

Performance vs. Reject Option: As a complementary analysis, Fig. 6 shows the accuracy, F-measure, and G-mean (first, second, and third row of plots, respectively) of both the best DL approach and shallow classifier vs. the censoring threshold γ on the three considered datasets. All the plots include, for a complete comparison, the CR vs. γ. This analysis delves into the possibility for DL architectures to classify apps more accurately only from reliably-labeled biflows. We notice that a threshold value implying different performance w.r.t unclassified samples can be theoretically observed only if $\gamma \geq 1/L$ (recall that L denotes the number of classes). This corresponds to ≈ 0.02 in the case of Android and iOS datasets, whereas this value equals 0.5 for the FB/FBM dataset. Results show that all the methods globally benefit from increasing γ at the price of a decreasing ratio of classified instances. However, only in the multi-class dataset it is evident a relevant performance improvement with a negligible ratio of unclassified samples, whereas for the FB/FBM (binary) dataset this trend is sharper and less advantageous (although the best DL classifier tends to be "less wrong" than its shallow counterpart, while having almost the same CR vs. γ profile). Since the marginal gain of DL classifiers w.r.t. shallow counterparts can be observed over all the γ range, we can infer that more sophisticated DL architectures (and more informative inputs) would be needed for an accurate classification. Specifically, by rejecting the classification of only 10% of instances, in the case of Android dataset, the 1D-CNN (L7-784) is able to achieve $\geq 90\%$ accuracy, $\geq 85\%$ F-measure, and $\geq 90\%$ G-Mean. Similarly, for the iOS dataset, the 2D-CNN (L7-784) achieves $84 \div 88\%$, scores with the same CR. Unluckily, in the FB/FBM case, achieving the same target performance would require $\geq 40\%$ biflows to be censored. This result again underlines the

[12]The times refer to the same hardware architecture ($8 \times$ Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz with Ubuntu 16.04 (64 bit)) in the same load conditions (i.e. the DL classifier is the sole CPU-intensive running process).

[13]For example, in our scenario, the observed RTPE for the pre-training phase (of five AE layers) equals $16.97\ (\pm 0.27)$ s in Android, $11.35\ (\pm 0.08)$ s in iOS, and $9.21\ (\pm 0.15)$ s in FB/FBM case.

[14]We omit, for brevity, the performance with "DIR-N_p" input, as it has been shown to be unable to reach satisfactory performance and its behavior with varying N_p can be qualitatively inferred from "MAT-N_p" results.



(a) LSTM + 2D-CNN [23]. (b) LSTM + 2D-CNN [23]. (c) LSTM + 2D-CNN [23].

(d) 1D-CNN [21]. (e) 2D-CNN [20]. (f) 1D-CNN [21].

Figure 5: Performance of the best DL-based classifier fed with **"MAT-N_p" input** (top row) and **"L7-N" input** (bottom row): Accuracy [%], F-measure [%], G-mean [%] (left axis), and RTPE [s] (right axis) vs. **first N_p packets** (top row) and **first N bytes** (bottom row), for the **Android (a, d)**, **iOS (b, e)**, and **FB/FBM (c, f)** datasets. Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

DL framework *limitations in tackling an "overlapped-apps" classification task* with the present input/architecture choices.

V. Lessons Learned and Challenges

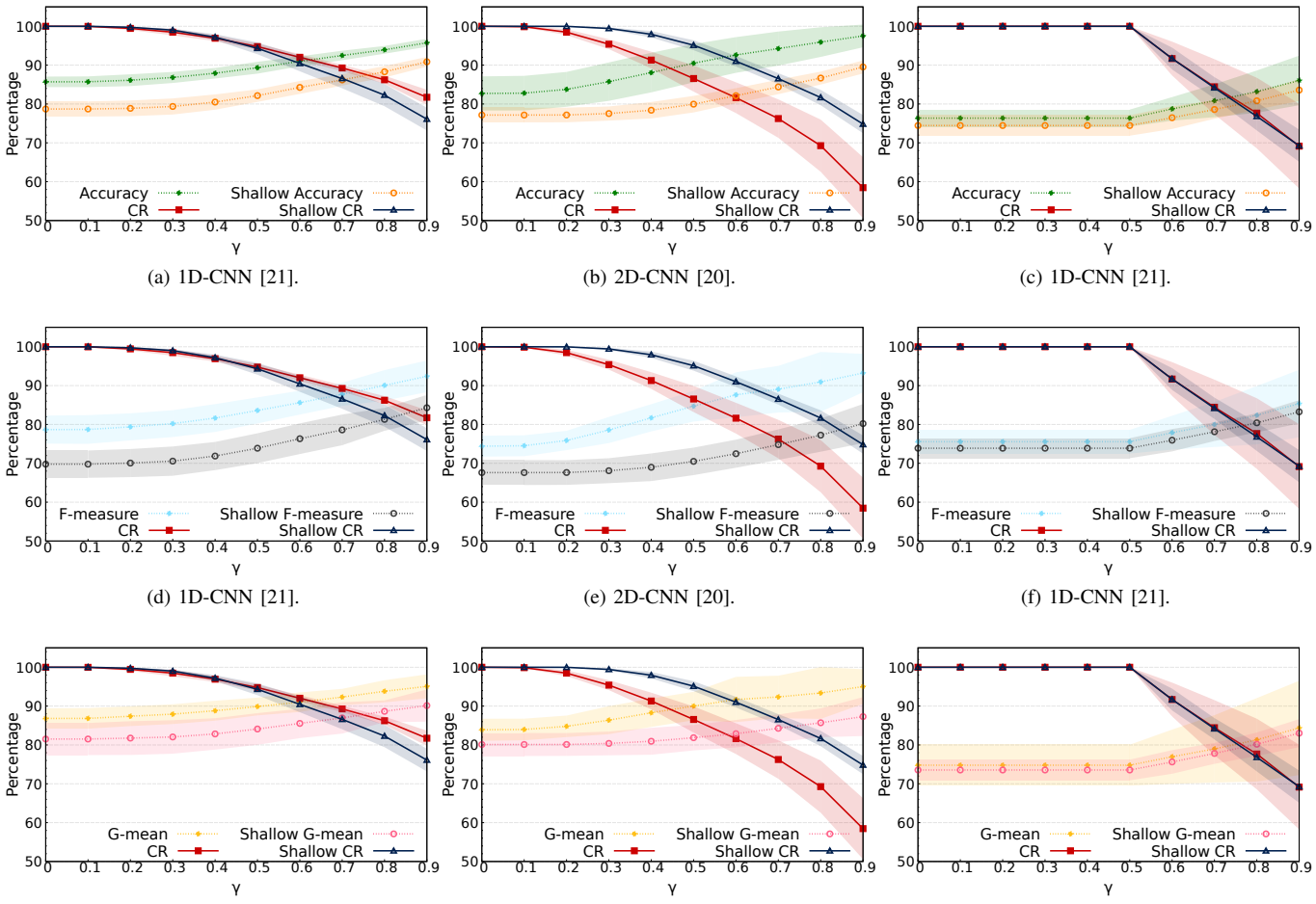
We tackled TC of mobile (encrypted) traffic via a DL approach for the first time in the literature. Our work provided not only a wide experimental analysis based on a newly-developed framework for comprehensive evaluation and comparison (Fig. 1) obtained by dissecting existing DL works in standard TC, but also the vital groundwork for *sound* advances on the general encrypted TC topic. Precisely, this analysis has enabled the surfacing of a list of guidelines and sparks, and highlights caveats of traffic analysis domain, so as to avoid pitfalls in the design and evaluation of DL-based (mobile) traffic classifiers and be the springboard of real-world implementations [44]. Hereinafter we summarize our conclusions as *lessons learned*, each with corresponding open *challenges*.

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 to its formalization*. Recent literature has ascertained that a naïve accuracy comparison is not sufficient, and measures reflecting a per-app behavior (F-measure, G-mean, confusion matrices, etc.) are increasingly considered [10, 16], given the high app number potentially involved in the classification task. Going further, we investigated DL architectures output at a finer detail by means of Top-K accuracy and by providing a performance analysis with a reject option, being essential in highly multi-instance and multi-class classification tasks, respectively, such as the mobile one [10].

This analysis was also enriched with a training-phase complexity evaluation of DL architectures (via the defined RTPE). Indeed, although test complexity is directly associated to the classifier at run-time, training complexity equally represents a key aspect in mobile TC, where frequent re-training of a classifier is required, due to aging of training data because of apps/OS updates [10, 30]. For completeness, the framework included a baseline "shallow" network to assess DL (classification) performance gain and a state-of-the-art ML-based classifier [13], using handcrafted and flow-based features.

The lack of a comprehensive and principled approach to DL-based classifiers applied to TC has been the main motivation to this work. This challenge is specifically important in research on TC as it is affected by the lack of up-to-date human-generated public datasets. This can be mainly attributed to the difficulty of anonymizing traffic traces in ways that both do not significantly affect the information useful for classification, and preserve users privacy in the face of future de-anonymization attacks. This issue is further worsened for mobile traffic, where the possibility of sharing significant and up-to-date datasets is hindered by both the highest privacy concerns and fast-paced evolution of traffic mix. Hence, an agreed-upon and comprehensive approach to comparison is vital to the progress of knowledge in this field. With this work we highlight this challenge, and provide a first response to it.

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 (or in a set of packets) likely leads to *misleading performance results*. One relevant case is [20, 21], adopting the "ALL layers" input, and thus overlooking the presence of PCAP metadata. Similarly, the input proposed in [23] includes port numbers, yielding DL statistical port-based architectures. Furthermore, whether destination port may be useful in some



(a) 1D-CNN [21]. (b) 2D-CNN [20]. (c) 1D-CNN [21].

(d) 1D-CNN [21]. (e) 2D-CNN [20]. (f) 1D-CNN [21].

(g) 1D-CNN [21]. (h) 2D-CNN [20]. (i) 1D-CNN [21].

Figure 6: Accuracy (a-c), F-measure (d-f), G-mean (g-i), and ratio of classified samples (CR) [%] vs. **censoring threshold** γ of the best DL-based classifier, fed with **"L7-784" input**, for the **Android (a, d, g)**, **iOS (b, c, h)**, and **FB/FBM (c, f, i)** datasets. Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

"static" contexts, this is never the case for the source port, which is subject to a choice depending on sequential numbering or, in a more sophisticated fashion, to randomization. On the other hand, the directions of packets belonging to a biflow (albeit representing an unbiased input type) were shown to be not informative enough as in the case of WF [24, 33, 34]. Therefore, a key outcome of this study was to *skim informative and unbiased information from traffic data* to be used as DL classifiers' input. Finally, since the complexity of DL-based traffic classifiers directly depends on the size of the input data, we preliminary investigated the "minimum required" size for each type of input for an accurate classification. Results have shown that, whether the fields of the first 16 to 20 packets are usually sufficient to reach the highest performance reported with "MAT-N_p" input, a clear trend is not evident for payload input "L7-N". Accordingly, this motivates a deeper investigation, also in terms of a more effective representation of payload (i.e. byte-based or at a higher/lower resolution).

Associated with this lesson learned, we surface the challenge of carefully analyzing and selecting the input of DL algorithms. Unluckily an elaborated input selection process contrasts one of the main promises of DL approaches, i.e. the reduced need of domain expertise. Indeed, this process potentially limits the generality of the obtained solution. In the case of DL-based classifiers, this issue is worsened by the black-box nature of most algorithms, as the performance impact of specific inputs is barely or not-at-all predictable. Hence, striking the right balance between naive application and expertise-driven effort constitutes a still open challenge.

Choice of TC object: This work, for brevity and consistency with surveyed DL-based traffic classifiers, only considered biflow-based TC, given the higher performance experienced w.r.t. its flow-based counterpart [20, 21]. However, recent mobile TC literature has shown the appeal of TC objects exploiting the bursty traffic nature (namely, the "service burst") [10, 13, 16, 25]. Although appealing, a definition of reasonable (and effective) input data for the latter TC object is not as straightforward as in the case of (bi)flows given the presence of a varying number of biflows toward the same destination IP/port. Moreover, while there is longstanding practical experience and mature technology working with biflows, using classification results from service bursts becomes hard to translate into actionable and sensible reactions. Therefore, this aspect deserves further attention and research in our opinion.

Fine-grained design of DL traffic classifiers: Results in Sec. IV-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 based on payload data which is, by definition, one-dimensional). Moreover, from our analysis of the literature we found that the tuning of hyper-parameters of DL algorithms is substantially overlooked (just tentative values are provided, if at all).

From these observations we derive that, given the heterogeneous information available from traffic data, the need for *advanced hybrid DL architectures* arises. Also, though DL architectures relieve the designer from the feature design issue, they come with many hyper-parameters to be tuned (e.g. the optimizer, the number of layers/hidden nodes, the regularizers). To explore the performance gain brought by fine-grained design, this further process can be as complex and resource-demanding as feature design. On the plus side, differently from feature design this process can be automated, as it is less domain-driven.

Further challenges posed by DL in the field of TC pertain to the training dataset. Indeed, 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 used 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).

We conclude confirming that DL algorithms applied to mobile TC indeed constitute a promising approach, but the current state of research on this application has yet to reach the maturity level of DL in other fields. In this work we have systematically explored this aspect and provided guidelines and directions to face the challenges that we surfaced.

REFERENCES

[1] A. Dainotti, A. Pescapè, and K. C. Claffy. Issues and future directions in traffic classification. *IEEE Network*, 26(1), 2012.

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

[3] D. Rajashekar, N. Zincir-Heywood, and M. Heywood. Smart phone user behaviour characterization based on autoencoders and self organizing maps. In *IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016.

[4] Y. Fu, J. Liu, X. Li, and H. Xiong. A multi-label multi-view learning framework for in-app service usage analysis. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(4), 2018.

[5] Sandvine. Global Internet Phenomena Spotlight: Encrypted Internet Traffic., 2016.

[6] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill. Studying TLS usage in Android apps. In *13th ACM CoNEXT*, 2017.

[7] G. Aceto, A. Dainotti, W. De Donato, and A. Pescapè. PortLoad: taking the best of two worlds in traffic classification. In *IEEE Conference on Computer Communications (INFOCOM) Workshops*, 2010.

[8] 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 21st International Conference on Mobile Computing and Networking (MobiCom)*, 2015.

[9] 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 Workshop on Offensive Technologies (WOOT)*, 2016.

[10] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1), 2018.

[11] V. Carela-Español, P. Barlet-Ros, M. Solé-Simó, A. Dainotti, W. de Donato, and A. Pescapè. K-dimensional trees for continuous traffic classification. In *TMA 2010, Zurich, Switzerland*, 2010.

[12] Y.-D. Lin, C.-N. Lu, Y.-C. Lai, W.-H. Peng, and P.-C. Lin. Application classification using packet size distribution and port association. *Journal of Network and Computer Applications*, 32(5), 2009.

[13] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.

[14] A. Hajjar, J. Khalife, and J. Díaz-Verdejo. Network traffic application identification based on message size analysis. *Journal of Network and Computer Applications*, 58, 2015.

[15] A. Dainotti, F. Gargiulo, L. I. Kuncheva, A. Pescapé, and C. Sansone. Identification of traffic flows hiding behind TCP port 80. In *2010 IEEE International Conference on Communications*, May 2010.

[16] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè. Multi-classification approaches for classifying mobile app traffic. *Journal of Network and Computer Applications*, 103, 2018.

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

[18] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè. Mobile encrypted traffic classification using deep learning. In *IEEE/ACM Network Traffic Measurement and Analysis Conference (TMA)*, 2018.

[19] Z. Wang. The Applications of Deep Learning on Traffic Identification., 2015.

[20] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng. Malware traffic classification using convolutional neural network for representation learning. In *IEEE International Conference on Information Networking*, 2017.

[21] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *IEEE International Conference on Intelligence and Security*

Informatics (ISI), 2017.

[22] M. Lotfollahi, R. Shirali, M. J. Siavoshani, and M. Saberian. Deep packet: a novel approach for encrypted traffic classification using Deep Learning. *arXiv*, 2017.

[23] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access*, 5, 2017.

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

[25] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are? smartphone fingerprinting via application behaviour. In *ACM WISEC*, 2013.

[26] 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 Conference on Communications and Network Security (CNS)*, 2015.

[27] J. Kampeas, A. Cohen, and O. Gurewitz. Traffic classification based on zero-length packets. *IEEE Transactions on Network and Service Management*, 15(3), Sept 2018.

[28] K. Shahbar and A. N. Zincir-Heywood. Packet momentum for identification of anonymity networks. *Journal of Cyber Security and Mobility*, 6(1), 2017.

[29] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Trans. Inf. Forensics Security*, 11(1), 2016.

[30] H. F. Alan and J. Kaur. Can Android applications be identified using only TCP/IP headers of their launch time traffic? In *9th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, 2016.

[31] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial Naïve-Bayes classifier. In *ACM workshop on Cloud computing security*, 2009.

[32] M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *ACM 13th conference on Computer and communications security (CCS)*, 2006.

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

[34] P. Sirinam, M. Imani, M. Juarez, and M. Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. *arXiv*, 2018.

[35] C. Zhang, X. Wang, F. Li, Q. He, and M. Huang. Deep learning–based network application classification for SDN. *Wiley Transactions on Emerging Telecommunications Technologies*, 2018.

[36] H. Huang, H. Deng, J. Chen, L. Han, and W. Wang. Automatic multi-task learning system for abnormal network traffic detection. *Int. Journal of Emerging Technologies in Learning*, 13(4), 2018.

[37] Y.-C. Chen, Y.-J. Li, A. Tseng, and T. Lin. Deep learning for malicious flow detection. In *IEEE 28th International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017.

[38] H. Shi, H. Li, D. Zhang, C. Cheng, and X. Cao. An efficient feature generation approach based on deep learning and feature selection techniques for traffic classification. *Computer Networks*, 2018.

[39] L. Vu, C. T. Bui, and Q. U. Nguyen. A deep learning based method for handling imbalanced problem in network traffic classification. In *ACM SoICT*, 2017.

[40] D. Li, Y. Zhu, and W. Lin. Traffic identification of mobile apps based on variational autoencoder network. In *13th IEEE International Conference on Computational Intelligence and Security (CIS)*, 2017.

[41] G. D. Gil, A. H. Lashkari, M. Mamun, and A. A. Ghorbani. Characterization of encrypted and VPN traffic using time-related features. In *2nd International Conference on Information Systems Security and Privacy*, 2016.

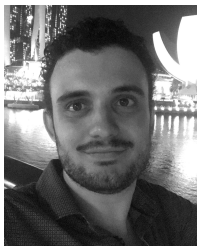
[42] F. Chollet et al. Keras. https://keras.io, 2015.

[43] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *ACM CoNEXT*, 2006.

[44] W. De Donato, A. Pescapé, and A. Dainotti. Traffic identification engine: an open platform for traffic classification. *IEEE Network*, 28(2), 2014.

Giuseppe Aceto is an Assistant Professor at University of Napoli Federico II. He has a PhD in telecommunication engineering from the same University. His work falls in monitoring of network performance and security (focusing on censorship) both in traditional and SDN network environments. He is also working on bioinformatics and ICTs applied to health. He is the recipient of a best paper award at IEEE ISCC 2010, and 2018 Best Journal Paper Award by IEEE CSIM.

Domenico Ciuonzo (S'11-M'14-SM'16) is Assistant Professor at University of Napoli Federico II, Italy. He holds a Ph.D. in Electronic Engineering from University of Campania "L. Vanvitelli", Italy and, from 2011, he has held several visiting researcher appointments. Since 2014 he is editor of several IEEE, IET and ELSEVIER journals. His research interests include data fusion, statistical signal processing, wireless sensor networks, traffic analysis and machine learning.



Antonio Montieri is a PhD Student at the Department of Electrical Engineering and Information Technology of the University of Napoli Federico II since 2017. He has received his MS Degree from the same University in 2015. His work is focused on network measurements, (encrypted and mobile) traffic classification and modeling, monitoring of cloud network performance. Antonio has co-authored 15 papers and 5 posters accepted for publication in international journals and conference proceedings.

Antonio Pescapè (SM'09) is a Full Professor of computer engineering at the University of Napoli Federico II. His work focuses on Internet technologies and specifically on measurement, monitoring, and analysis of the Internet. He has co-authored more than 200 papers and is the recipient of a number of awards. He is involved in several research projects on Internet Technologies and he is reviewer and evaluator of research projects for international agencies, governments, and EU commission.