Mobile Traffic Classification through Physical Control Channel Fingerprinting: a Deep Learning Approach

Hoang Duy Trinh*, Angel Fernandez Gambin†, Lorenza Giupponi*, Michele Rossi† and Paolo Dini*

***Abstract*—The automatic classification of applications and services is an invaluable feature for new generation mobile networks. Here, we propose and validate algorithms to perform this task, at *runtime*, from the *raw physical control channel* of an *operative mobile network*, without having to decode and/or decrypt the transmitted flows. Towards this, we decode Downlink Control Information (DCI) messages carried within the LTE Physical Downlink Control CHannel (PDCCH). DCI messages are sent by the radio cell in clear text and, in this paper, are utilized to classify the applications and services executed at the connected mobile terminals. Two datasets are collected through a large measurement campaign: one labeled, used to train the classification algorithms, and one unlabeled, collected from four radio cells in the metropolitan area of Barcelona, in Spain. Among other approaches, our Convolutional Neural Network (CNN) classifier provides the highest classification accuracy of 98%. The CNN classifier is then augmented with the capability of rejecting sessions whose patterns do not conform to those learned during the training phase, and is subsequently utilized to attain a fine grained decomposition of the traffic for the four monitored radio cells, in an *online* and *unsupervised* fashion.**

***Index Terms*—Traffic Classification, Traffic Modeling, Mobile Networks, LTE, 5G, Machine Learning, Neural Networks, Deep Learning, Data Analytics.**

I. Introduction

WIRELESS mobile technology is advancing at a fast pace, through better monitor resolutions, larger memories, higher communication speeds, a higher number of connected devices, etc., and, with that, more requirements in terms of supported data rates [1], [2], new services, and a higher network responsiveness across diverse physical contexts [3]. As mobile systems become more complex, network operators attempt to transform their architecture through new functionalities and procedures including security, reliability and enhanced service management. Traffic classification is necessary in this context to prioritize and/or protect certain flows, to prevent the injection of malicious data, and to allocate the needed network resources to serve the traffic generated by the end users.

A large body of work exists in the area of mobile traffic classification (see Section VI for an in depth discussion of the related work). The key challenge of existing classification algorithms consists in the identification, and in the subsequent computation, of a number of *representative features*. These features are then used to train algorithms that classify the data flows at runtime. Most of the surveyed approaches leverage some *domain knowledge*, which is utilized to *manually* obtain the feature set, i.e., crafted by a skilled human expert. However, the use of deep learning techniques has recently paved the way to automatic feature discovery and extraction, often leading to superior performance. For example, in [4] encrypted traffic is categorized through deep learning architectures, proving their better performance with respect to shallow neural network classifiers. The authors of [5] present a mobile traffic super-resolution technique to infer narrowly localized traffic consumption from coarse measurements: a deep-learning architecture combining Zipper Network (ZipNet) and Generative Adversarial neural Network (GAN) models is proposed to accurately reconstruct spatio-temporal traffic dynamics from measurements taken at low resolution. In [6], the identification of mobile apps is carried out by automatically extracting features from labeled packets through Convolutional Neural Networks (CNNs), which are trained using raw Hypertext Transfer Protocol (HTTP) requests, achieving a high classification accuracy. We stress that the work in these papers, as the majority of the other techniques discussed in Section VI, use statistical features obtained from application or Internet Protocol (IP) level information for both service and app identification, along with UDP/TCP port numbers.

The solution here presented sharply differs from previous approaches. In fact, it accurately classify mobile traffic from *radio-link* data by solely processing the information coming from the *control channel*, without requiring any prior knowledge and without having to decode and/or decrypt the transmitted data flows. Specifically, we design and evaluate, via proof-of-concept implementations, non-intrusive tools for the online estimation of Long Term Evolution (LTE) cellular activity, i.e., the type of traffic that users exchange with their serving base station. As we quantify, our technology allows one to infer *with high accuracy* the service (e.g., audio-streaming, video-streaming, video-conferencing) and the application (e.g., Skype, Vimeo, You Tube, etc.) that are being used by the connected mobile users. This is accomplished by decoding LTE Physical Downlink Control CHannel (PDCCH) messages (i.e., radio-link level data), which are transmitted in clear text without having to break any security protocol (encryption). To do this, we leverage OWL [7], a tool that allows decoding the Downlink Control Information (DCI) messages carried in the LTE PDCCH, where control information is exchanged between the LTE Base Station (eNodeB) and the connected

*CTTC/CERCA, Av. Carl Friedrich Gauss, 7, 08860, Castelldefels, Barcelona, Spain {hoangduy.trinh, lorenza.giupponi, paolo.dini}@cttc.es,
†DEI, University of Padova, Via G. Gradenigo, 6/B, 35131 Padova, Italy. {afgambin, rossi}@dei.unipd.it.

This work has received funding from the European Union Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 675891 (SCAVENGE), by the Spanish Government under project TEC2017-88373-R (5G-REFINE) and has been supported, in part, by MIUR (Italian Ministry of Education, University and Research) through the initiative "Departments of Excellence" (Law 232/2016).

User Equipments (UEs). From such messages, radio-link level settings for the user communication, e.g., modulation and coding scheme, transport block size, allocated resource blocks, etc., are obtained.

From DCI data, two datasets are created:

1) a *labeled dataset*, used to *train* different service and app classification algorithms, where labeling is made possible by injecting traffic generated by a mobile terminal under our control into the network;

2) an *unlabeled dataset*, used for *traffic profiling* purposes, which is populated by monitoring, for a full month, mobile traffic from four operative radio cell sites with different demographic characteristics within the metropolitan area of Barcelona, in Spain.

For the traffic analysis, the focus is put on a few selected services and applications that dominate the radio resource usage, but the approach can be readily extended to other scenarios. Raw DCI data is used directly as input into deep learning classifiers (automatic feature extraction), achieving accuracies as high as 98% for both mobile service and app identification tasks. Moreover, an original technique to use our best classifier in *unsupervised settings* is presented, to profile the mobile traffic from operative radio cell sites at runtime. The developed classification algorithms, as well as our experimental results, are highly novel within the traffic monitoring literature, which only provides hourly and *aggregated* measures for typical days [8] [9], and where traffic profiling is performed from UDP/TCP, IP or above IP flows, e.g., [4]–[6].

We believe that the approach proposed in this work brings a high value along several dimensions, as discussed next. Collecting and processing control channel information may reduce the storage capabilities of the network elements, since the volume of DCI messages is much smaller than that from the user plane. Moreover, consumer privacy is maintained as no deep packet inspection mechanism on user content is required. The proposed solution works directly at the network edge, so any action to attain a better and secure management of the network (e.g., network slicing optimization) can be promptly taken. Classification time does not depend on the number of transmitted user data packets, but on the internal system clock (i.e., the frequency of DCI messages in our case). Our tool permits a better understanding of spectrum needs across time and space. Note that there are limited means to non-intrusively monitor user density and traffic demand in real-time. These measurements are key for the correct dimensioning of future mobile systems, and the investigation of new data communication and processing techniques for wireless channel interfaces. We only found a limited number of datasets available including mobile data traffic [10] [11], but they are often incomplete, poorly documented and contain aggregated data, often hourly averaged. The proposed technique allows the extraction of data flows with a granularity of one second, tagging the type of service and application exploited by each mobile user within an LTE cell.

In addition to the above, we believe that the proposed approach brings a considerable added value to the research community in general. In fact, we propose a methodology for cellular networks control data generation, which can be reproduced by anyone interested in getting data. This kind of raw data is hardly released by operators, especially with the high level of granularity that we have been able to retrieve. This approach allows any researcher without access to industrial data, to carry out research, by relying on real cellular data, in a key area like Artificial Intelligence (AI)-enabled Beyond 5G networks, which is also aligned with the Next Generation Self-Organized Networks Next Generation Self Organized Networks (NG-SON) vision promoted by, e.g., the 3GPP and the Next Generation Mobile Networks (NGMN) alliance of mobile network operators. The data on which we have worked is extracted from an LTE network, but the methodology can be reproduced, adapting the underlying software, also to New Radio (NR), as the PDCCH flows are transmitted unencrypted also in 5G NR [12] [13].

In summary, the original contributions of this work are:

- *Mobile Data Labeling at the Edge of the Network:* we collect and label LTE PDCCH DCI data traces from six mobile apps to create a unique correspondence between the software programs (the apps) and the session identifiers that were assigned to them by the eNodeB. The result is a *labeled dataset* of real DCI data from selected applications, i.e., YouTube, Vimeo, Spotify, Google Music, Skype and WhatsApp video calls.
- *Classification and Benchmarks:* we tailor deep artificial Neural Networks (NNs), namely Multi-Layer Perceptron (MLP), Recurrent Neural Networks (RNNs) and CNNs, to perform classification tasks for mobile services and app identification on the labeled dataset. Moreover, we compare their performance against a number of benchmark state-of-the-art classifiers.
- *Mobile Data Collection at the radio-link level from an Operative Mobile Network:* we collect real LTE PDCCH DCI data traces, with a time granularity of 1 ms, from four eNodeBs located in the metropolitan area of Barcelona, in Spain. Each of these datasets has a duration of 1 month.
- *Mobile Service Profiling from Unlabeled Data:* the CNN classifier, which is found to be the best among all Neural Network (NN) schemes, is augmented with the capability of rejecting *out of distribution* sessions, i.e., sessions whose statistical behavior departs from those learned during the training phase. This makes it possible to use it with unlabeled traffic, as an online and unsupervised traffic monitoring tool. The augmented CNN classifier rejects those sessions for which it is uncertain, providing a robust classification outcome. Through its use, the four selected eNodeBs are monitored, getting a fine grained traffic decomposition.

The paper is organized as follows. Section II presents the experimental framework and the proposed methodology to obtain the two datasets. Section III introduces the two classification problems, namely, service and app identification and presents the classification algorithms. The performance of the classification algorithms is assessed in Section IV. In Section V, the CNN classifier is augmented with the capability of rejecting out of distribution sessions. Thus, the mobile

traffic from four selected cell sites of an operative mobile network in Spain is decomposed over a full day. The related work on mobile traffic classification is reviewed in Section VI, and some concluding remarks are provided in Section VII.

II. Dataset Creation

Fig. 1 shows the different building blocks of the experimental framework that has been developed to populate the unlabeled and labeled datasets. Briefly, the *data measurement and collection* block acquires data from the LTE PDCCH channel to extract the relevant DCI information. *Data preparation*, instead, processes the gathered DCI data so that it can be used for training and classification purposes.

A. Data Collection System

In LTE, the eNodeB communicates scheduling information to the connected UEs through the DCI messages that are carried within the PDCCH with a time granularity of 1 ms. While the actual user content is sent over *encrypted dedicated channels*, i.e., the Physical Uplink/Downlink Shared Channel (PUSCH/PDSCH respectively), the **PDCCH is transmitted in clear text** and can be decoded. To process DCI data, we have adapted the OWL monitoring tool [7]. A Software-Defined Radio (SDR) has been programmed, acquiring the PDCCH via an open-source software sitting on top of the srs-LTE library [14], which makes it possible to synchronize and monitor the channel over a specified LTE bandwidth. The SDR is connected to a PC that performs the actual decoding of DCI data: in our experimental settings, we used a low cost Nuand BladeRF x40 SDR and an Intel mini-NUC, equipped with an i5 2.7 Ghz multi-core processor, 256 GB Solid State Storage (SSD) storage and 18 GB of RAM.

Decoded DCI messages for a connected UE contain the following scheduling information [15]:

- Radio Network Temporary Identifier (RNTI),
- Resource Block (RB) assignment,
- Modulation and Coding Scheme (MCS).

DCI messages use RNTIs to specify their destination. RNTIs are 16-bit identifiers that are employed to address UEs in an LTE cell. They are used for different purposes such as to broadcast system information (SI-RNTI), to page a specific UE (P-RNTI), to carry out a random access procedure (RA-RNTI), and to identify a connected user, i.e., the cell RNTI (C-RNTI). In this work, we are interested in the C-RNTI, that is temporarily assigned when the UE is in RRC (Radio Resource Control) CONNECTED state, to uniquely identify it inside the cell. The C-RNTI can take any unreserved value in the range [0x003D–FFF3]. Once the C-RNTI is assigned to a connected UE, the DCI information directed to this terminal is sent using this C-RNTI, which is transmitted in clear text as part of the PDCCH channel. Hence, knowing the C-RNTI allows tracking a specific connected user within the radio cell. Assuming that the C-RNTI is known (see Section II-C), the following information about the ongoing communication for this UE can be extracted from its DCI data:

- *Number of allotted resource blocks:* in LTE, a RB represents the smallest resource unit that can be allocated to any user. The number of resource blocks that are assigned to a UE (N_{RB}), is derived based on the DCI bitmap.
- *Modulation order and code rate:* the MCS is a 5-bit field that determines the modulation order and the code rate that are used, at the physical layer, for the transmission of data to the UE.
- *Transport Block Size (TBS):* the TBS specifies the length of the packet to be sent to the UE in the current Transmission Time Interval (TTI). It is derived by from a lookup table by using MCS and N_{RB}, see [15].

The rationale behind this work is that the (unencrypted) downlink and uplink TBS data of a given UE should provide sufficient information for learning algorithms to reliably classify the app and the service that the user is running.

B. Unlabeled Dataset

Thanks to the just described DCI collection system, four cell sites of a Spanish mobile network operator in the metropolitan area of Barcelona have been monitored for a full month. The selected eNodeBs are located in areas having different demographic characteristics and land uses, so as to diversify the captured traffic in terms of service and app behavior. We have named the datasets according to the corresponding neighborhood: *PobleSec* (mainly residential area), *Born* (mixed residential, transport and leisure area), *Castelldefels* (mixed suburban and campus area), *Camp Nou* (mixed residential and stadium area). In total, we have collected more than 68 GB of DCI data from the LTE PDCCH. Fig. 2 shows the locations of the four monitored sites, along with that of the data collection system. After the data collection, the signaling associated with each active C-RNTI is extracted from the PDCCH DCI data stream, and is prepared for the classifier. During this, we discarded short-length traces, which are mainly due to signaling, paging and background traffic. These accounted for less than 3% of the total traffic in the monitored radio cells.

C. Labeled Dataset

A *labeled* dataset is obtained by running specific services and apps at a mobile terminal under our control, detecting its C-RNTI within the PDCCH channel and finally associating the corresponding DCI trace with a *label*, which links it to the service/app that is executed at the UE. The mobile device used to generate the labeled dataset is a Huawei Y6 phone running an Android operating system. YouTube, Vimeo, Spotify, Google Music, Skype and WhatsApp video calls have been utilized to generate traffic. For video calls, a background video was run to generate the video and the audio. Music and videos were selected at random from automatically generated thematic lists.

Generating data sessions is easy, and boils down to running a specific app from a device that we control, and that is connected to the monitored eNodeB. The difficult part is to identify the generated data flow among those carried by the PDCCH channel, which contains DCI information for *all* the connected UEs within the radio cell. We made this labeling possible by injecting a *watermark* into the traffic that we generated by the controlled UE, so that it could be easily identified among all other users.

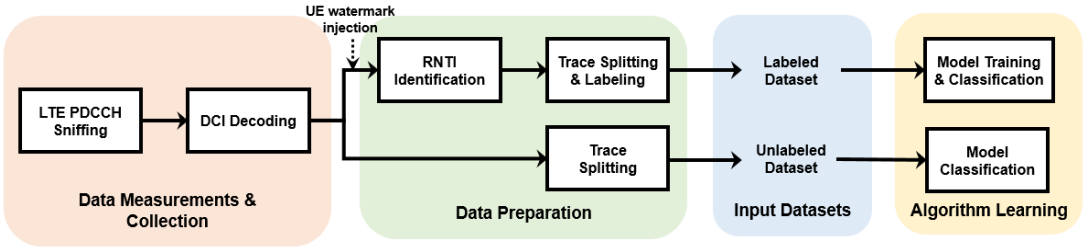
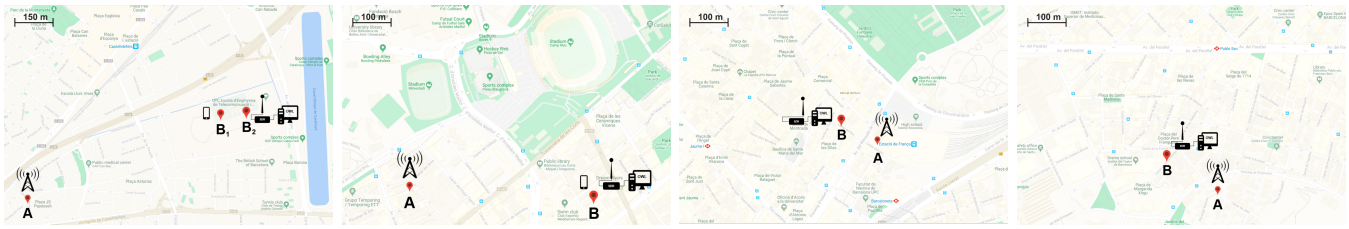


Fig. 1: Experimental framework adopted for the creation of the unlabeled and labeled datasets.



(a) Castelldefels: suburban area with a university campus.

(b) Camp Nou: mainly residential area with Barcelona FC stadium.

(c) Born: mixed residential, transport and leisure area.

(d) PobleSec: mainly residential area.

Fig. 2: Maps of Barcelona metropolitan areas where the measurement campaign took place for the creation of unlabeled and labeled datasets. In the maps, the eNodeB location is denoted by A, whereas the data collection system and the mobile terminal are marked as B. In Castelldefels, the mobile terminal has been placed in two different locations (B_1 and B_2).

1) Data preparation and watermarking

The data preparation procedure is divided into two phases: **1)** the *identification* of the C-RNTI corresponding to the controlled UE, **2)** the *extraction* and *labeling* of the corresponding DCI trace. In the LTE PDCCH channel, each UE is identified by the C-RNTI, which uniquely identifies the mobile terminal within the radio cell. This identifier is *temporary*, i.e., it changes after short inactivity periods. This is done to prevent the plain tracking of mobile users, since the PDCCH is sent unencrypted. To allow traffic labeling (i.e., user identification), we introduce a *watermark* into the traffic generated by our mobile terminal. This watermark amounts to producing, for each application, a regular pattern: any instance of a given application (e.g., YouTube) is run for a pre-defined amount of time (80 seconds in our measurements), then, a pause interval of fixed duration is inserted before running another instance of the app for further 80 seconds. We loop this over time, obtaining a duty cycled activity pattern that is easily distinguishable from all the other activity traces within the radio cell. Through this watermarking procedure, we can successfully associate our UE with the corresponding C-RNTI from the DCI. Also, we split the traces into different sessions (the difference instance of the same app running for 80 seconds) thanks to the duty cycled pattern, where subsequent sessions are separated by the pause interval (of fixed duration). The *label*, corresponding to the application that is executed at the mobile terminal, is finally associated with the extracted DCI data. We remark that this procedure makes it possible to capture several instances of a given app in a row, which are stored in our dataset with the associated app/service label and does not modify the normal network behavior to serve the running app. Also, we found instances of 80 seconds (observation time) to be more than sufficient to classify the service/app and longer instances would lead to negligible improvements.

In our measurement campaign, we have recorded and labeled $M = 11,601$ mobile sessions, gathering the scheduling information contained in the DCI messages for selected apps. We considered three data-intensive services: *video streaming*, *audio streaming* and *real-time video calling*, which represent classes producing a considerable amount of traffic and taking most of the network resources [1]. For each service type, we chose two popular applications: *Spotify* and *Google Music* for audio, *YouTube* and *Vimeo* for video streaming, while for the video calling we picked two instant-messaging applications, namely, *Skype* and *WhatsApp Messenger*.

A large measurement campaign was conducted to expose the mobile terminal running the selected apps to different *radio link conditions*, thus obtaining a comprehensive dataset. In particular, the UE was placed into two different locations (termed B_1 and B_2 in Fig. 2a) within the Castelldefels radio cell to experience different received signal qualities (-84 dBm and -94 dBm for B_1 and B_2, respectively), and in the Camp Nou eNodeB during football matches, to capture data in high cell load conditions.

Fig. 3 shows a few radio resource usage patterns collected for the selected apps. Some similarities can be recognized within the same service class. For example, audio and video streaming present similar behaviors. Also, significant differences can be observed between the radio resource usage of real time video calls (*Skype* and *WhatApp Video*) and the other apps. Video and audio streaming applications use up a high amount of radio resources at the beginning of the sessions, buffering most of the content into the terminal memory. Real time video calling, instead, entails a continuous transmission and a more constant usage of radio resources throughout the sessions. Note that the amount of data exchanged in the uplink direction is significant only for this service class, since a video call requires a bidirectional communication.

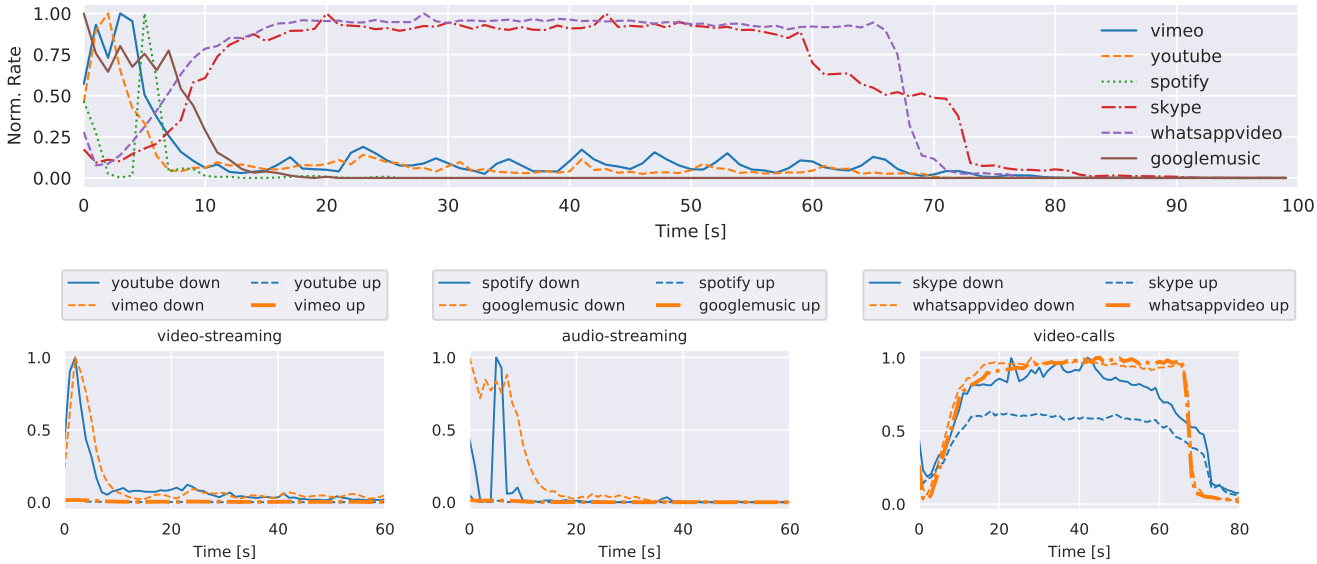


Fig. 3: Traffic pattern snapshots showing the normalized data rate for different applications as a function of time.

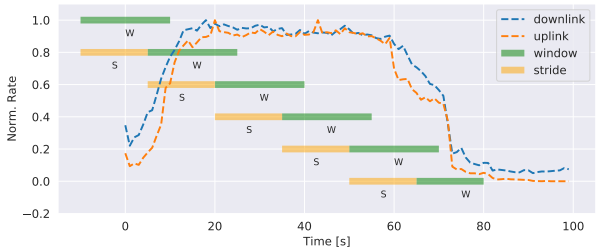


Fig. 4: Sliding window of 20s length and 15s stride, applied to a sample video-streaming session.

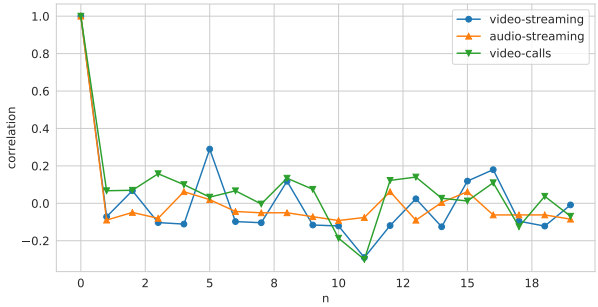


Fig. 5: Pearson correlation between the initial and the following sessions running in the controlled UE.

D. Synchronous and Asynchronous Sessions

Through the watermarking approach and the splitting procedure, we obtained a labeled dataset, where each session, depending on the service, presents patterns similar to those shown in Fig. 3. Assuming that the beginning and the end of each session are known is rather optimistic, as in a real measurement setup we have no means to accurately track these instants. Put it another way, it is unlikely that the LTE PDCCH measurements and the application run on the UE will be temporally *synchronized*. Synchronizing the measurement with the beginning of each session would facilitate the classification task, since most of the generated traffic is buffered on the terminal at the beginning, see Fig. 3, and this behavior is a distinctive feature that is easy to discriminate.

To ensure the applicability of our classifiers to real world (asynchronous) cases, we account for *asynchronous* sessions, entailing that the classification algorithm has no knowledge about the instants where the sessions start and finish. Specifically, each session is split using a sliding window of length W seconds, moved rightwards from the beginning of the session with a stride of S seconds, see Fig. 4. The split sessions (asynchronous sessions), of W seconds each, represent the input data to our classification algorithms. Note that W and S are hyper-parameters of the proposed classification frameworks.

E. Sessions Correlation over Time

As a sanity check, we verify the soundness of the watermarking strategy: our aim is to understand whether the transmission of user data in the form of duty-cycled patterns may affect the way in which the eNodeB handles the communication from our terminal, e.g., through some advanced channel reservation mechanism. In that case, in fact, our watermarking strategy would be of little use, as it would introduce scheduling artifacts that do not occur in real life conditions. To verify this, we evaluated the Pearson correlation between the initial session (i.e., when the UE connects to the LTE PDCCH for the first time and it is assigned a new C-RNTI) and the following ones. Fig. 5 shows that, for each of the three services, the correlation is high only when we compare the first session with itself ($n = 0$). Instead, low values are observed between the first session and the following ones ($n > 1$), indicating that the behavior of the eNodeB scheduler is not affected by the repetitive actions (i.e., the duty-cycled activity) performed at the UE side.

III. Classification Problem

A. Problem Definition

Let $M = 11,601$ be the total number of sessions obtained through the data preparation procedure of Section II-C, $L = 80$ seconds is the duration of each session, and $D = 2$ is the number of communication directions (downlink and uplink). We define $\boldsymbol{X}$ as the input dataset tensor with size $M \times L \times D$, where the m-th row vector $\boldsymbol{x}_m$ contains the trace associated with $W \in [0, L]$ TBS samples per session for both downlink

and uplink directions. The time-series described by $\boldsymbol{x}_m$ is the input sequence of our algorithms.

A classifier estimates a function $c : \boldsymbol{X} \rightarrow \boldsymbol{Y}$, where the output matrix $\boldsymbol{Y}$ has size $M \times K$, with K representing the number of classes. The row vector $\boldsymbol{y_m} = c(\boldsymbol{x_m}) = [y_{m1}, \ldots, y_{mK}]$ contains the probabilities that session m belongs to each of the K classes, with $\sum_k y_{mk} = 1$. The final output of the classifier is class $k^\star$, where $k^\star = \text{argmax}_k(y_{mk})$. The following classification objectives are addressed:

O1) **Service identification:** to classify the collected sessions into $K = 3$ classes, namely, *audio streaming*, *video streaming* and *video calls*;

O2) **App identification:** to identify which app is run at the UE. In this case, the number of output classes is $K = 6$, namely, *Spotify*, *Google Music*, *YouTube*, *Vimeo*, *Skype* and *WhatsApp Messenger*.

Next, we present the considered classification algorithms, grouping them into two categories: those based on artificial neural networks and those based on standard machine learning techniques (referred to here as benchmark classifiers).

B. Deep Neural Networks

Next, we describe how we tailored three neural network architectures to solve the above traffic classification problem, namely, Multilayer Perceptron (MLP), Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs).

1) Multilayer Perceptron

A multilayer perceptron is a feedforward and fully-connected neural network architecture. The term "feed-forward" refers to the fact that the information flows in one direction, from the input to the output layer. An MLP is composed of, at least, three layers of nodes: an input, a hidden and an output layer. A directed graph connects the input with the output layer and each neuron in the graph uses a non-linear activation function to produce its output. Links are weighted and the backpropagation algorithm is utilized to train the network in a supervised fashion, i.e., to find the set of network weights that minimize a certain error function, given an input set of examples and the corresponding labels. For further details, see [16].

The MLP that we use for mobile traffic classification has *three* fully connected layers. The first layer MLP_1 contains $N_{MLP_1} = 128$ neurons, the second layer MLP_2 has $N_{MLP_2} = 64$ neurons and the third layer MLP_3 is fully connected, with K neurons and a softmax activation function to produce the final output. The output of MLP_3 is the class probability vector $\boldsymbol{y_m}$.

All neurons in layers MLP_1 and MLP_2 use a leaky version of the Rectified Linear Unit (ReLU) (*leaky ReLU*) activation function. Leaky ReLUs help solve the vanishing gradient problem, i.e., the fact that the error gradients that are backpropagated during the training of the network weights may become very small (zero in the worst case), preventing the correct (gradient based) adaptation of the weights. To prevent this from happening, leaky ReLUs have a small negative slope for negative values of their argument [17]. To train the presented MLP architecture, we use the *RMSprop* gradient descent algorithm [18], by minimizing the *categorical cross-entropy loss function* $L(\boldsymbol{w})$, defined as [16]

$$L(\boldsymbol{w}) = -\sum_{\boldsymbol{x}_m \in \boldsymbol{B}} \sum_{k=1}^{K} t_k(\boldsymbol{x}_m) \log(y_{mk}(\boldsymbol{w}, \boldsymbol{x}_m)). \quad (1)$$

where $\boldsymbol{t(x)_m} = [t_1(\boldsymbol{x}_m), \ldots, t_k(\boldsymbol{x}_m)]$ contains the class labels associated with the input trace $\boldsymbol{x}_m$, i.e., $t_k = 1$ if $\boldsymbol{x}_m$ belongs to class k and $t_k = 0$ otherwise (1-of-K coding scheme). Vector $\boldsymbol{w}$ contains the MLP weights and $y_{mk}(\boldsymbol{w}, \boldsymbol{x}_m)$ is the MLP output obtained for input $\boldsymbol{x}_m$. Eq. (1) is iteratively minimized using the training examples in the batch set $\boldsymbol{B} \subset \boldsymbol{X}$, where $\boldsymbol{B}$ is changed at every iteration so as to span the entire input set $\boldsymbol{X}$.

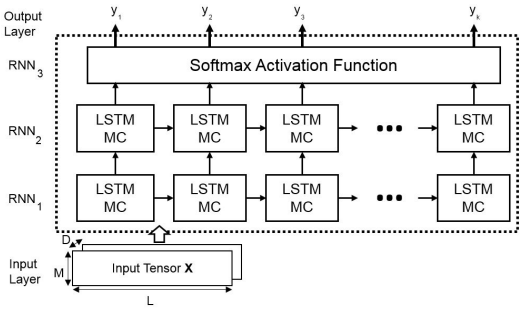


Fig. 6: RNN architecture.

2) Recurrent Neural Networks

Recurrent Neural Networks (RNNs) have been conceived to extract features from temporal (and correlated) data sequences. Long Short-Term Memory (LSTM) networks are a particular type of RNN, introduced in [19]. They are capable of tracking long-term dependencies into the input time series, while solving the vanishing-gradient problem that affects standard RNNs [20].

The capability of learning long-term dependencies is due to the structure of the LSTM cells, which incorporates gates that regulate the learning process. The neurons in the hidden layers of an LSTM are Memory Cells (MCs). A MC has the ability to retain or forget information about past input values (whose effect is stored into the cell states) by using structures called *gates*, which consist of a cascade of a neuron with sigmoidal activation function and a pointwise multiplication block. Thanks to this architecture, the output of each memory cell possibly depends on the entire sequence of past states, making LSTMs suitable for processing time series with long time dependencies [19]. The input gate of a memory cell is a neuron with sigmoidal activation function (σ). Its output determines the fraction of the MC input that is fed to the cell state block. Similarly, the forget gate processes the information that is recurrently fed back into the cell state block. The output gate, instead, determines the fraction of the cell state output that is to be used as the output of the MC, at each time step. Gate neurons usually have sigmoidal activation functions (σ), while the hyperbolic tangent ($\tanh$) activation function is usually adopted to process the input and for the cell state. All the internal connections of the MC have unitary weight [19].

The proposed RNN based traffic classification architecture is shown in Fig. 6. In our design, we consider three stacked layers

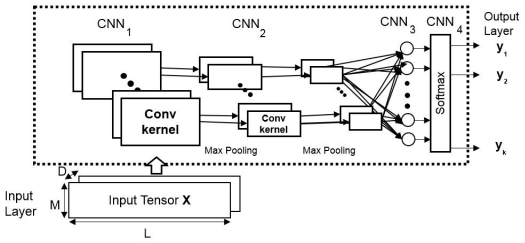


Fig. 7: CNN architecture.

combining two LSTM layers and a final fully connected output layer. The first and the second layer (respectively RNN_1 and RNN_2) have $N_{RNN_1} = N_{RNN_2} = 180$ memory cells. The fully connected layer RNN_3 uses the softmax activation function and its output consists of the class probability estimates, as described in Section III-B1.

3) Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are feed-forward deep neural networks differing from fully connected MLP for the presence of one or more convolutional layers. At each convolutional layer, a number of kernels is used. Each kernel is composed of a number of weights and is convolved across the entire input signal. Note that the kernel acts as a filter whose weights are re-used (*shared weights*) across the entire input and this makes the network connectivity structure sparse, i.e., a small set of parameters (the kernel weights) suffices to map the input into the output. This leads to a considerably reduced computational complexity with respect to fully connected feed forward neural networks, and to a smaller memory footprint. For more details the reader is referred to [21].

CNNs have been proven to be excellent feature extractors for images and inertial signals [22] and here we show their effectiveness for the classification of mobile traffic data. The CNN architecture that we designed to this purpose is shown in Fig. 7. It is a 1-Dimensional CNN with two main parts: the first four layers perform convolutions and max pooling in cascade, the last two layers are fully connected. The first convolutional layer CNN_1 uses one dimensional kernels (1×5 samples) performing a first filtering of the input and processing each input vector (rows of $\boldsymbol{X}$) separately. The activation functions are linear and the number of convolutional kernels is $N_{CNN_1} = 32$. The second convolutional layer, CNN_2, uses one dimensional kernels (1×5 samples) with non-linear hyperbolic tangents as activation functions, and the number of convolutional kernels is $N_{CNN_2} = 64$. Max pooling is separately applied to the outputs of CNN_1 and CNN_2 to reduce their dimensionality and increase the spatial invariance of features [22]. In both cases, a one-dimensional pooling with a kernel of size 1×3 is performed. A third fully connected layer, CNN_3, performs dimensionality reduction and has $N_{CNN_3} = 32$ neurons with Leaky ReLU activation functions. This layer is used in place of a further convolutional layer to reduce the computation time, with a negligible loss in accuracy. The last (output) layer CNN_4 is fully connected with softmax activation functions, and returns the class probability estimates, see Sections III-B1.

TABLE I: Configuration parameters for the benchmark classifiers.

Algorithm	Parameters	Note - Reference
Linear SVM	• *penalty* = L2 • *loss* = Hinge Loss • $c = 0.025$	• c: penalty parameter for the error term • extended to multi-class with one-vs-rest [23]
Logistic Regressor	• *penalty* = L2 • $c = 1$	• c: inverse of the regularization strength • extended to multi-class with one-vs-rest [23]
Nearest Neighbours	• $K = 3$ • $p = 2$ • *metric* = Minkowski	• K: number of neighbors for queries • p: distance metric parameter • $p = 2$ amounts to using the Euclidean distance [24]
Random Forest	• *n. estimators* $= 10$ • *max depth* $= 5$ • *criterion* = entropy	• *n. estimators*: number of trees in the forest • *max depth*: maximum depth of a tree • *criterion*: function to measure the quality of a split of subsets [26]
Gaussian Processes	• *kernel* = RBF • σ = Logistic func. • *approx.* = Laplacian	• Radial Basis Function (RBF) used as kernel • σ is the sigmoid function used to "squash" the nuisance function • Laplacian method used to approximate the non Gaussian Posterior [27]

C. Benchmark classifiers

Other standard classification schemes have been tailored to the considered tasks O1 and O2. The selected algorithms are: *Linear Logistic Regression*, *K-Nearest Neighbours* and *Linear SVM*, as examples of linear classifiers; *Random Forest*, as an ensemble learning method, and *Gaussian Processes* as an instance of Bayesian approaches. The implementations of Linear Logistic Regression, K-Nearest Neighbours and Linear SVM are based on [23], [24] and [25], respectively. The Random Forest implementation is based on [26], whereas for the classifier based on Gaussian Processes we refer to [27]. Configuration parameters and implementation details for the benchmark classifiers are provided in Table I.

IV. Supervised Training and Comparison of Traffic Classification Algorithms

The performance tests have been carried out using an Intel core i7 machine, with 32 GB of RAM and an NVIDIA GTX 980 GPU card. We divided the dataset, featuring 11601 labeled DCI sessions, into training and validation sets with a split ratio of 70% - 30%. These sets are balanced, as they contain the same percentage of traces for all classes. The classification algorithms have been implemented in Python. We have used `keras` library on top of Tensorflow backend for the implementation of deep NNs. For the benchmark classifiers, we used the popular `sklearn` library.

A. Performance Metrics

The classification performance is assessed through the following metrics:

1) **Accuracy**: defined as the ratio between the number of correctly classified sessions to the total number of sessions.
2) **Precision** P: defined, for each class, as the ratio between

Algorithm	Precision	Recall	F-Score	# Parameters	Accuracy Sync%	Accuracy Async %	Difference %
Linear SVM	0.811	0.812	0.805	36726	81.23	68.41	-12.8
Logistic Regressor	0.806	0.816	0.809	486	81.61	65.72	-15.9
Nearest Neighbours	0.843	0.845	0.841	36720	84.51	79.65	-4.9
Random Forest	0.821	0.835	0.827	41310	83.52	70.21	-13.3
Gaussian Processes	0.874	0.871	0.871	146720	87.43	81.21	-6.2
Neural Networks							
MLP	0.900	0.900	0.900	19014	90.04	84.61	-5.4
RNN	0.967	0.968	0.968	392046	96.57	92.93	-3.6
CNN	0.978	0.976	0.977	25062	97.77	93.20	-4.5

TABLE II: Classifiers comparison for the app identification task.

Algorithm	Precision	Recall	F-Score	# Parameters	Accuracy Sync%	Accuracy Async %	Difference %
Linear SVM	0.908	0.908	0.907	19843	90.80	79.61	-11.2
Logistic Regressor	0.904	0.904	0.904	243	90.42	81.11	-9.3
Nearest Neighbours	0.925	0.925	0.925	19840	92.76	83.45	-9.3
Random Forest	0.915	0.915	0.915	22320	91.57	84.25	-7.3
Gaussian Processes	0.932	0.928	0.929	73360	93.21	82.51	-10.7
Neural Networks							
MLP	0.943	0.939	0.942	18819	94.31	93.38	-0.9
RNN	0.981	0.982	0.981	391503	98.21	95.38	-2.8
CNN	0.986	0.988	0.988	24963	98.87	95.40	-3.5

TABLE III: Classifiers comparison for the service identification task.

true positives T_p and the sum between true positives and false positives F_p,

$$P = \frac{T_p}{T_p + F_p}, \tag{2}$$

3) **Recall** R: defined, for each class, as the ratio between the true positives T_p and the sum of true positives and false negatives F_n,

$$R = \frac{T_p}{T_p + F_n}. \tag{3}$$

4) **F-Score** F is defined as the harmonic mean of precision P and recall R,

$$F = \left(\frac{\frac{1}{P} + \frac{1}{R}}{2} \right)^{-1} = 2\frac{RP}{R+P}. \tag{4}$$

Note that the definition of precision and recall only applies to classification tasks with one class. However, tasks O1 and O2 both have a number of classes $K > 2$, namely, $K = \{3, 6\}$ for app and service identification, respectively. Thus, precision and recall are separately calculated for all the K classes, and their average is shown in the following numerical results.

B. Comparison of Classification Algorithms

1) Accuracy and Algorithm Training

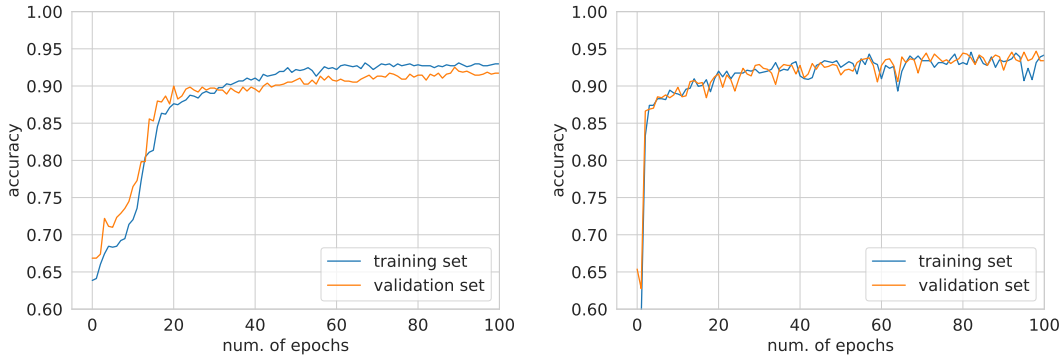
Tables II and III summarize the obtained performance metrics for the deep NNs and the benchmark classifiers for app and service identification, respectively. Results refer to an observation window $W = 80$. In both synchronous and asynchronous cases, higher accuracy is achieved by deep NNs than the benchmarks (up to $+13.8\%$ on app identification, $+8.7\%$ on service classification). The algorithm based on Gaussian Processes performs the best among the benchmark classifiers. In general, the higher the complexity (i.e., the number of parameters, and also the number of hidden layers for NNs), the higher the performance. The only exception to this is provided by CNNs, which present the highest accuracy but use a small number of parameters. This fact confirms the high efficiency of convolution operations in processing high amount of data with complex temporal structure, and the effectiveness of CNN parameter sharing. CNNs requires only 6% of the variables used up by RNNs, achieving a better accuracy. This also translates into a faster training: in Fig. 8, we show the accuracy as a function of the number of epochs for training and validation for RNNs and CNNs, using an observation window of $W = 80$ TBS samples. The number of epochs required by the CNNs to reach an accuracy higher than 90% is fewer than 20 (Fig. 8b), whereas for RNNs convergence is achieved only after 30 epochs (Fig. 8a).

From Tables II and III, we observe a significant performance gap between the service and app classification tasks: the difference in the accuracy is higher than 8% for the benchmark classifiers and ranges from 2 to 6% for deep NNs. This is mainly due to the higher number of classes for the app identification task, which increases the mis-classification probability, as discussed shortly below in the analysis of the Confusion Matrix of Fig. 9.

Results to assess the effect of asynchronous readings are shown in Tables II, III and Fig. 10. As shown in the last two columns of Tables II and III, training the algorithms with asynchronous sessions decreases their classification accuracy. We observe a general decrease for all the algorithms (-6.0% for service identification, -7.7% for app identification, on average). This occurs as the beginning of the sessions holds key information on the session type, thus simplifying the classification task (as shown in Fig. 3). However, for both classification problems, neural network-based approaches are more robust to asynchronous readings, showing a performance degradation of -4.3%, while the degradation for the benchmark algorithms is -8.4%, see also Fig. 10.

2) Confusion Matrix

A deeper look at the performance of CNNs is provided by the confusion matrices of Fig. 9, whose rows and columns



(a) RNN. (b) CNN.

Fig. 8: Accuracy *vs* number of epochs for training and validation sets for the app identification task in the asynchronous case.

respectively represent true and predicted labels, and all values are normalized between 0 and 1. For the service classification task (Fig. 9a), CNNs only misclassify the video streaming sessions: 2% of those are labeled as video calls. For the app identification task (Fig. 9b), errors (4%) mainly occur for Skype and WhatsApp videocalls. These errors are understandable, as these are both interactive real-time video applications and, as such, their traffic patterns bear similarities. The lowest performance is found for Vimeo traces, for which 88% of the sessions are correctly classified. Here our CNN-based classifier confuses them with the other video applications for both streaming service (Youtube - 3%) and real-time calling (WhatsApp and Skype - 6% and 3%, repectively).

3) Impact of Different Window Sizes

Fig. 11 shows the classification accuracy as a function of the window size, W. Results here are shown for 40 epochs training. For the app identification task, 40 seconds suffice for CNNs and RNNs to reach accuracies higher than 90%, with negligible additional improvements for longer observation periods. Periods shorter than 40 seconds provide less accurate results. Similar trends are observed for the service classification task. However, in this case after 20 seconds the accuracy of CNNs and RNNs is already higher than 90%, due to the smaller number of classes. In summary, the ability of CNNs and RNNs to extract representative statistical features from a session grows with the input data length. In our tasks, deep NN algorithms become very effective as monitoring intervals get longer than 20 seconds.

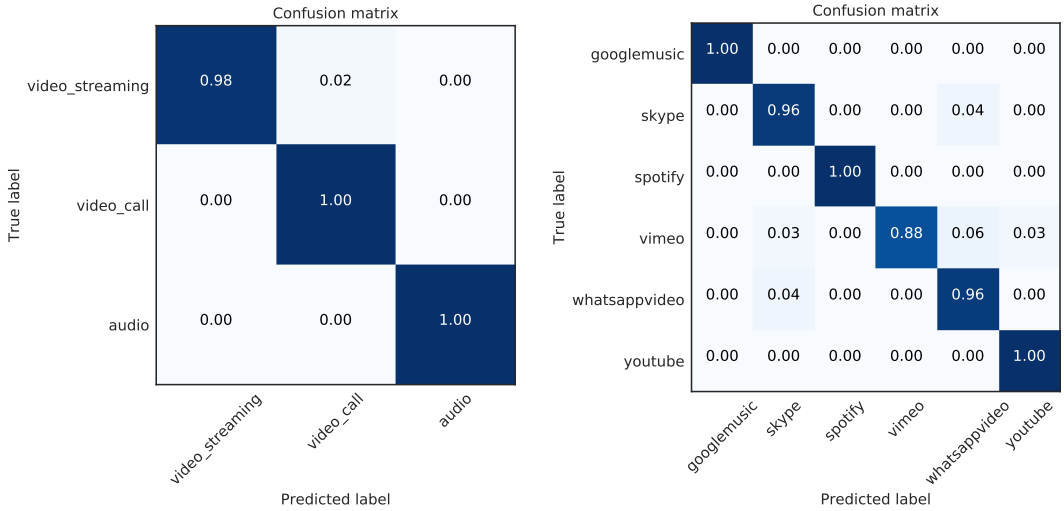
Contrarily to W, we have experimentally verified that parameter S does not appreciably affect the performance of the algorithm.

V. Unsupervised Traffic Profiling

Next, we analyze the mobile traffic exchanged within the four selected cell sites (see Section II-B). The traffic load is modeled in terms of aggregated traffic dynamics and type of service requests over the 24 hours of a day. The identification of mobile traffic, for each of the considered services, is performed using the trained classifiers from Section III with the *unlabeled* dataset. Formally, for each eNodeB, the corresponding unlabeled dataset is stored into the tensor $\boldsymbol{X}'$, with size $M' \times N \times D$, where M' corresponds to the number of monitored RNTIs (sessions, N to the number of collected samples per session and $D = 2$ is the number of communications directions (downlink and uplink). Given $\boldsymbol{X}'$, as input, the classifier c computes the output $\boldsymbol{Y}'$, whose analysis provides a detailed characterization of the mobile user requests for the eNodeB within the monitored time span. Vectors $\boldsymbol{x}'_m$ and $\boldsymbol{y}'_m = c(\boldsymbol{x}'_m)$ respectively indicate the m-th sample of $\boldsymbol{X}'$ and $\boldsymbol{Y}'$. In this paper, we restrict our attention to the unsupervised classification of services, and use the CNN classifier, as it yields the highest accuracy.

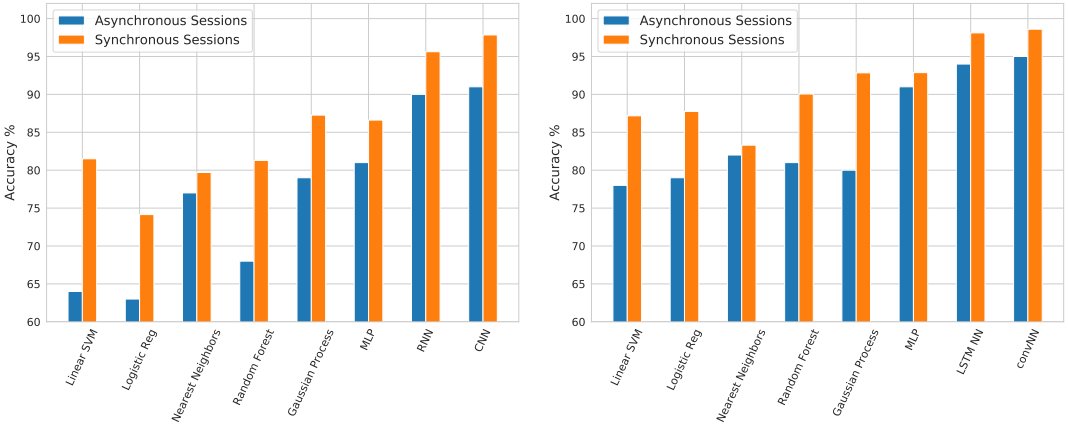
A. Aggregated Traffic Analysis

Fig. 12 shows the *aggregated* traffic demand for the four selected eNodeBs over the 24 hours of a typical day, where each curve has been normalized with respect to the maximum traffic peak occurred during the day for the corresponding eNodeB. The four traffic profiles have a different trend, which depends on the characteristics of the served area (demographics, predominant land use, etc.), as confirmed by [28]. **PobleSec** is a residential neighborhood and, as such, presents traffic peaks during the evening, at 5 and 11 pm. **Born** is instead a downtown district with a mixed residential, transport and leisure land use. Two peaks are detected: the highest is at lunchtime around 2 pm, whereas the second one is at dinner time, from 9 pm. This traffic behavior is likely due to the many restaurants and bars in the area. **CampNou** is mainly residential and presents a similar profile to PobleSec. However, Barcelona FC stadium is located in this area, and three football matches took place during the monitored period (events started at 8:45 pm and ended at 10:45 pm). As expected, a higher traffic intensity is observed during the football match hours. In particular, we registered a high amount of traffic exchanged between 7 pm to 1 am, i.e., before, during and after the events. This behavior is probably due to the movement of people attending the matches. **Castelldefels** is a suburban and low populated area with a university campus. The traffic variation suggests a typical office profile with traffic peaks at 10 am and 5 pm. However, in this radio cell the amount of traffic exchanged is the lowest observed across all eNodeB sites, i.e., 6.8 Gb/hour in the peak hours. The highest traffic intensity was measured in Camp Nou, reaching a peak of 106.1 Gb/hour (29.5 Mb/s on average). Intermediate peak values are detected in Poble Sec and Born, amounting to 49.7 Mb/s and 46.1 Mb/s, respectively. The only common pattern among the four areas is the low traffic intensity at night, between 2 am and 7 am.



(a) Confusion matrix for service identification. (b) Confusion matrix for app identification.

Fig. 9: Confusion matrices for the CNN algorithm.



(a) App identification. (b) Service identification.

Fig. 10: Impact of synchronizing the DCI readings with the start of the user's sessions.

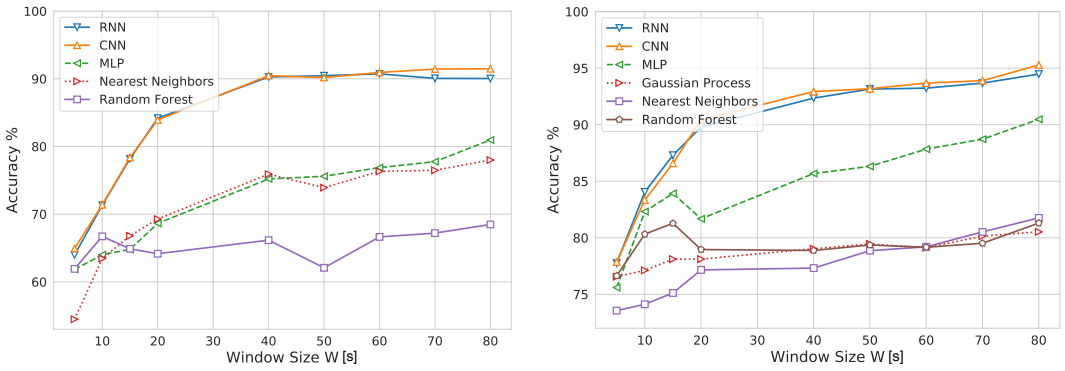
B. Traffic Decomposition at Service Level

The set of applications that we have labeled is restricted to those apps and services that dominate the radio resource usage. However, additional apps may also be present in the monitored traffic, such as Facebook, Instagram, Snapchat, etc. These apps generate mixed content, including audio-streaming, video-streaming and video-calling. Additional service types may also be generated by, e.g., web-browsing and file downloading. While in the present work the classifiers were not trained to specifically track these apps, for a robust classification outcome, it is desirable that the audio and video streams that they generate will either be captured and classified into the correct service class, or at least flagged as unknown. To locate those traffic patterns for which our classifier may produce inaccurate results, in our analysis we additionally account for the detection of out-of-distribution (OOD) sessions, i.e., of DCI traces that show different traffic dynamics from those learned at training time. Other services, that are classified as OOD fall within the categories of Web browsing, file downloading, interactive applications such as texting and messaging. These are common but dot not take a large portion of the radio resources. Having an OOD class has the advantage of automatically detecting all of these, as an aggregate (marked as OOD), allowing the framework to attain higher accuracies for the remaining classes, which generate patterns that are similar to those that were observed during the learning phase.

To identify these "statistical outliers", the DCI data from each new session, $\boldsymbol{x}'_m$, is fed to the CNN and the corresponding softmax output vector $\boldsymbol{y}'_m = [y'_{m1}, \ldots, y'_{mK}]^T$ is used to discriminate whether $\boldsymbol{x}'_m$ is OOD or not, following the rationale in [29] [30]. In detail, the k-th softmax output corresponds to the probability estimate that a given input session $\boldsymbol{x}'_m$ belongs to class k, i.e., $y'_{mk} = \text{Prob}(\boldsymbol{x}'_m \in \text{class } k)$, with $k = 1, \ldots, K$. The classifier chooses the class $k^\star$ that maximizes this probability, i.e.,

$$k^\star = \underset{k}{\text{argmax}}\, y'_{mk}. \tag{5}$$

If a new app, not considered in the training phase, generates sessions having similar characteristics to those in the training set, namely, audio-streaming, video-streaming or real time video-calls, we expect the CNN to generalize well and return similar vectors at the output of the softmax layer. That is, the softmax vector that is outputted at runtime for the new app should be sufficiently "close" to the output learned by



(a) Accuracy for app identification.

(b) Accuracy vs Window Size W for service identification.

Fig. 11: Accuracy *vs* window lengths for app and service classification tasks in the asynchronous measurement case.

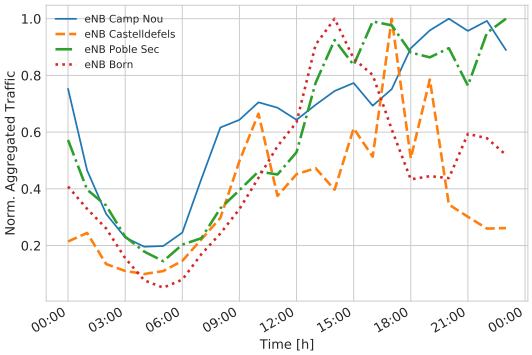


Fig. 12: Daily Aggregated Traffic for the four eNodeBs.

the classifier from the labeled dataset, as the new signal bears statistical similarities with those learned in the training phase. In this case, it makes sense to accept the session and classify it as belonging to class $k^\star$. Otherwise, the session would be classified as OOD.

The problem at hand, boils down to assessing whether the softmax output $\boldsymbol{y}'_m$ belongs to the statistical distribution learned by the CNN or it is an outlier. This amounts to performing outlier detection in a multivariate setting, with $\boldsymbol{y}'_m \in [0,1]^K$, $\sum_k y'_{mk} = 1$. Among the many algorithms that can be used to this purpose, we adopt the method based on Kernelized Spatial Depth (KSD) functions of [31] as it is lightweight and does not require the direct estimation of the probability density function (pdf) of the softmax output layer, which is a critical point, as good estimates require training over many points. Briefly, for a vector $\boldsymbol{y} \in \mathbb{R}^K$, we define the spatial sign function as $S(\boldsymbol{y}) = \boldsymbol{y}/\|\boldsymbol{y}\|$ if $\boldsymbol{y} \neq \boldsymbol{0}$ and $S(\boldsymbol{y}) = \boldsymbol{0}$ if $\boldsymbol{y} = \boldsymbol{0}$, where $\|\boldsymbol{y}\| = (\boldsymbol{y}^T\boldsymbol{y})^{1/2}$ is the norm-2. If $\mathcal{Y}_k$ is a training set containing ℓ softmax output vectors for a certain class k, $\mathcal{Y}_k = \{\boldsymbol{y}_1^{(k)}, \boldsymbol{y}_2^{(k)}, \ldots, \boldsymbol{y}_\ell^{(k)}\}$, the *sample spatial depth* associated with a new softmax output vector $\boldsymbol{y}'_m$ is:

$$D(\boldsymbol{y}'_m, \mathcal{Y}_k) = 1 - \frac{1}{|\mathcal{Y}_k \cup \boldsymbol{y}'_m| - 1} \left\| \sum_{\boldsymbol{y} \in \mathcal{Y}_k} S(\boldsymbol{y} - \boldsymbol{y}'_m) \right\|. \quad (6)$$

Note that $D(\boldsymbol{y}'_m, \mathcal{Y}_k) \in [0,1]$ provides a *measure of centrality* of the new point $\boldsymbol{y}'_m$ with respect to the points in the training set $\mathcal{Y}_k$. In particular, if $D(\boldsymbol{y}'_m, \mathcal{Y}_k) = 1$, it follows that $\| \sum_{\boldsymbol{y} \in \mathcal{Y}_k} S(\boldsymbol{y} - \boldsymbol{y}'_m) \| = 0$ and the new point is said to be the *spatial median* of set $\mathcal{Y}_k$, i.e., it can be thought of as the "center of mass" of this set. Hence, the spatial depth attains the highest value of 1 at the spatial median and decreases to zero as $\boldsymbol{y}'_m$ moves away from it. The spatial depth can thus be used as a measure of "extremeness" of a new data point with respect to a set. In [31], the spatial depth of Eq. (6) is *kernelized*, which means that distances are evaluated using a positive definite kernel map. A common choice, that we also use in this paper, is the generalized Gaussian kernel $\kappa(\boldsymbol{x}, \boldsymbol{y})$,

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-(\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{y})), \quad (7)$$

which provides a measure of similarity between $\boldsymbol{x}$ and $\boldsymbol{y}$. Noting that the square norm can be expressed as

$$\|\boldsymbol{x} - \boldsymbol{y}\|^2 = \boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{y}, \quad (8)$$

kernelizing the sample spatial depth amounts to expanding (6) and replacing the inner products with the kernel function κ. This returns the *sample KSD function* (Eq. (4) in [31]).

Session classification procedure in an unsupervised setting: the CNN classifier is augmented through the detection of OOD sessions, as follows:

- *Initialization:* for each class $k = 1, \ldots, K$ in the service/app identification task a number of softmax output vectors is computed by the *trained* CNN using the sessions in the training set. These softmax vectors are stored in the set $\mathcal{Y}_k$. Note that, being the results of a supervised training of the CNN, we know that the vectors in $\mathcal{Y}_k$ are all generated by a distribution that is correctly learned during the supervised training phase.
- *Feature extraction through the pre-trained CNN:* at runtime, as a new DCI vector $\boldsymbol{x}'_m$ is measured, it is inputted into the pre-trained CNN, obtaining the corresponding softmax output $\boldsymbol{y}'_m$.
- *Classification and OOD detection:* vector $\boldsymbol{y}'_m$ is used with Eq. (5) to assess the most probable class $k^\star$. At this point, Algorithm 1 of [31] is utilized to assess whether $\boldsymbol{y}'_m$ is an outlier. In case the vector is classified as an outlier, it is assigned to the OOD class, otherwise it is assigned to class $k^\star$.

Some final remarks are in order. The outlier detection algorithm uses a threshold $t \in [0,1]$, which allows exploring the tradeoff between *false alarm* rate and *detection* rate. Instead, the covariance matrix $\boldsymbol{\Sigma}$ controls the decision boundary for rejecting vectors, driving the tradeoff between

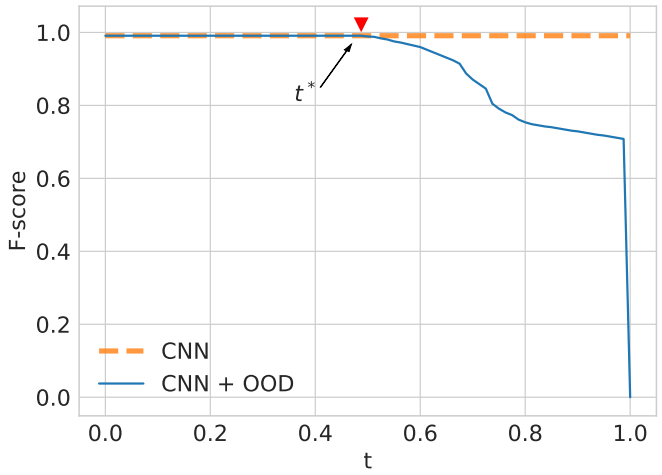


Fig. 13: Finding threshold $t^\star$ using the CNN with (solid line) and without (dashed line) the OOD detection mechanism.

the local and global behavior of KSD. If properly chosen, the contours of KSD should closely follow those of the (actual) underlying statistical distribution. $\mathbf{\Sigma}$ is learned, for each class k, from the training vectors in $\mathcal{Y}_k$, and for the following results we picked $\mathbf{\Sigma} = \mathbf{\Sigma}_2$ in [31].

Tuning the OOD threshold t: we define $\mathcal{S}_k$ as the set containing the training examples belonging to class k. We recall that $\mathcal{S}_k$ is used to compute the covariance matrix associated with the adopted Gaussian kernel, which models the contours of the pdf of the output softmax vectors. The threshold $t \in [0, 1]$ is instead used by the outlier detection algorithm to gauge the (kernelized) distance between the center of mass of set $\mathcal{S}_k$ and a new softmax vector, acquired at runtime. If $t = 1$, the kernelized spatial depth of the new point will always be smaller than or equal to t and all points will be rejected (marked as outliers). This is of course of no use. However, as we decrease t towards 0, we see that more and more points will be accepted, until, for $t = 0$, no rejections will occur. So, t determines the selectivity of the outlier detection mechanism, the higher t, the more selective the algorithm is. For our numerical evaluation, once the sets $\mathcal{S}_k$ are obtained for all classes k, we set this threshold by picking the highest value of it, $t^\star$, for which all the softmax vectors belonging to the test set are accepted, i.e., none of them is marked as an outlier (OOD). In other words, this is equivalent to making sure that the F-Score obtained over the test set from our trained CNN without the OOD mechanism enabled equals that of the CNN classifier augmented with the OOD detection capability. As $t^\star$ is the highest value of t for which all the data in the test set are correctly classified as valid, our approach amounts to tuning the threshold in such a way that the outlier detection algorithm will be as selective as possible, while correctly treating all the data in the test set. In Fig. 13, we show the F-Score as a function of t for the CNN algorithm with and without OOD detection. Threshold $t^\star = 0.48$ corresponds to the highest value of t for which the F-Score remains at its maximum, i.e., at the end of the flat region.

Experimental analysis of eNodeB traffic: in Fig. 14, the traffic decomposition into the considered service classes is shown for the four selected eNodeBs using $t^\star = 0.48$. The percentage of sessions identified as OOD, for which the classifier is uncertain, is also reported at the top of each bar. Common characteristics are observed in all the considered deployments:

- the most used service is video-streaming, with typical shares ranging from 50% to 80%. This confirms the measurements in [1] and [2].
- The least used service is video-call, whose share is typically between 5% and 10%, whereas audio-streaming takes 21% of the total traffic load.
- OOD sessions are consistently well below 8%. Note that this share accounts for all those apps that are not tracked by our classifier, such as texting, web browsing, and file transfers.

Through the proposed service identification approach, we can accurately characterize, at runtime, the used services. Moreover, the traffic decomposition at service level allows one to make some interesting considerations on the land use. For example, in a typical residential area (PobleSec) the audio-streaming service is the one used the least across the four monitored sites, with an average of 16.4%. Instead, in a typical office and university neighborhood (Castelldefels), audio-streaming has the highest traffic share across all sites (22% on average). Born and CampNou, which are two leisure districts, present a similar traffic distribution across the day. We finally remark that, while the traffic profiling results are shown using a time granularity of one hour, our classification tool allows for traffic decomposition at much shorter timescales, i.e., on a per-session basis.

VI. Related Work

The most common classification methods in the literature leverage the transport layer protocol, including UDP/TCP port analysis and/or packet inspection, since most Internet applications use UDP/TCP port numbers. For instance, the authors of [32] define a mobile traffic classifier as a collection of rules, including destination IP addresses and port numbers. Based on these rules, application-level mobile traffic identification is performed deploying a dedicated classification architecture within the network, and measurement agents at the mobile devices. However, port-based schemes hardly work in the presence of applications using dynamic port numbers [33].

A scheme based on deep packet inspection is presented in [34]. The authors of this article devise a technique for Code Division Multiple Access (CDMA) traffic classification, using correlation-based feature selection along with a decision tree classifier trained on a labeled dataset (which is labeled via deep packet inspection). The algorithm in [35] extracts application layer payload patterns, and performs maximum entropy-based IP-traffic classification exploiting different Machine Learning (ML) algorithms such as Naive Bayes, Support Vector Machines (SVMs) and partial decision trees. Remarkably, payload-based methods are limited by a significant complexity and computation load [33]. Furthermore, many mobile applications adopt encrypted data transmission due to security and

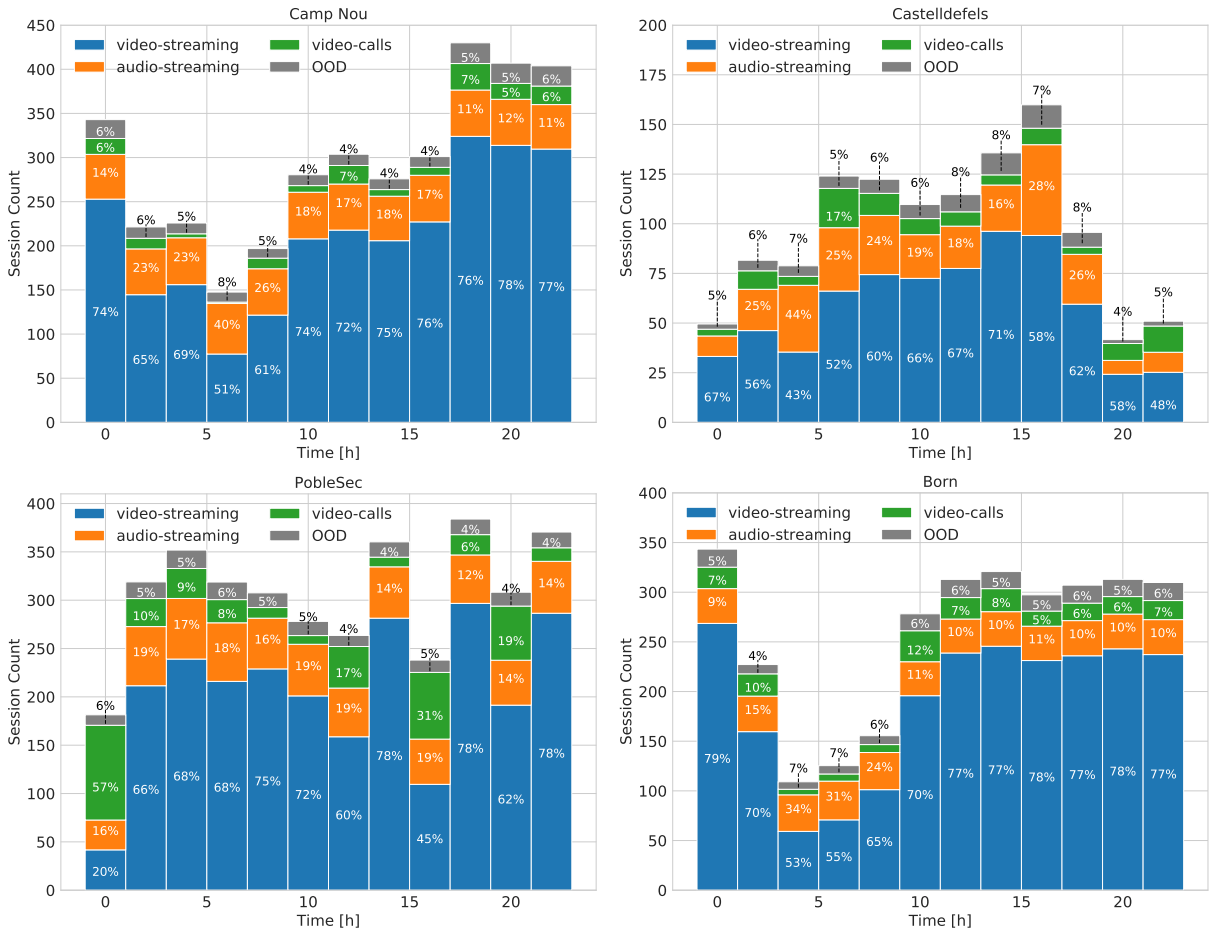


Fig. 14: Traffic decomposition at service level for the four monitored eNodeBs during the 24 hours of a day.

privacy concerns, which renders packet inspection approaches ineffective.

Recent works consider NNs [6], [5], [36]. The authors of [36] exploit the ability of deep NNs to perform classification of Android applications using system API-call sequences and investigate the effectiveness of NNs to learn complex features that can help in the malware detection task. In [6], mobile apps are identified by automatically extracting features from labeled packets through CNNs, which are trained using raw HTTP requests. In [4], encrypted traffic is classified using deep learning architectures (feed forward, convolutional and recurrent neural networks) for Android and iOS mobile apps, with and without exploiting TCP/UDP ports. The authors of [5] combine Zipper Networks (ZipNet) and Generative-Adversarial Networks (GAN) to infer narrowly localized and fine grained traffic generation from coarse measurements.

A systematic framework is devised in [37] for the comparison among different deep learning classifiers. Their performance is thoroughly investigated based on three mobile datasets of real human users activity, highlighting their drawbacks, discussing design guidelines and challenges. In [38], the same authors propose a multi-classification approach, where they combine the outputs of different classifiers in a modular way to improve the overall performance.

Several survey papers dealing with deep learning techniques applied to traffic classification can be found in [39], [40] and [41]. The authors of [39] provide general guidelines for classification tasks, present some deep learning techniques, show how they can be used for traffic classification and discuss open problems. The survey in [40] presents a deep learning-based framework for mobile encrypted traffic classification, reviewing existing work according to dataset selection, model input design, and model architecture, and highlighting open issues and challenges. Finally, a comprehensive review of the interplay between deep learning and mobile networking research is provided in [41], where the authors discuss how to tailor deep learning to mobile environments. Current challenges and open future research directions are also discussed.

We stress that most of the works in the literature, with the exception of [4]–[6] and [37], classify mobile traffic based on manual feature extraction and all the papers that we surveyed process network or application level data. Our work departs from previous research, as we classify mobile data gathered from the physical control channel at the network edge, at runtime, and without access to application data and TCP/UDP port numbers.

Although seemingly orthogonal to the traffic classification problem, which is the main focus of the present work, we believe it is appropriate to comment on the security implications of the developed technology. The central point here is that by just reading unencrypted control traffic, it is possible to perform network inference, achieving good results. One could for example understand the type of service and app that is being run by a certain user equipment, as we do in this paper,

which may be by itself a privacy breach. The authors of [42] go further and, using DCI information, show that it is possible to even identify and track smartphones, although we do not know the identity of the owner, a sort of *signature* of the way in which their smartphone interacts with the network over the wireless air interface can be computed, and such signature is enough to allow for its identification, with good accuracy. This is a privacy violation. The interested reader is invited to check the discussion in [42] and the references therein.

VII. Conclusions

In this paper, we have presented a framework that allows highly accurate classification of application and services from radio-link level data, at runtime, and without having to decrypt dedicated physical layer channels. To this end, we decoded the LTE Physical Downlink Control Channel (PDCCH), where Downlink Control Information (DCI) messages are sent in clear text. Through DCI data, it is possible to track the data flows exchanged between the serving cell and its active users, extracting features that allow the reliable identification of the apps/services that are being executed at the mobile terminals. For the classification of such traffic, we have tailored deep artificial Neural Networks NNs, namely, Multi-Layer Perceptron (MLP), Recurrent NNs and Convolutional NNs, comparing their performance against that of benchmark classifiers based on state-of-the-art supervised learning algorithms. Our numerical results show that NN architectures overcome the other approaches in terms of classification accuracy, with the best accuracy (as high as 98%) being achieved by CNNs. As a major contribution of this work, labeled and unlabeled datasets of DCI data from real radio cell deployments have been collected. The labeled dataset has been used to train and compare the classifiers. For the unlabeled dataset, we have augmented the CNN with the capability of detecting input DCI data that do not conform to that learned during the training phase: the corresponding patterns are detected and associated with an *unknown* class. This increases the robustness of the CNN classifier, allowing its use, at runtime, to perform fine grained traffic analysis from radio cell sites from an operative mobile network. To summarize, the main outcomes of our work are: **1)** a methodology to extract DCI data from the PDCCH channel, and for the use of such data to train traffic classifiers, **2)** the fine tuning and a thorough performance comparison of classification algorithms, **3)** the design of a novel technique for the *fine grained* and *online* traffic analysis of communication sessions from real radio cell sites, and **4)** the discussion of the traffic distribution resulting from such analysis from four selected sites of a Spanish mobile operator, in the city of Barcelona. As a future research direction, we foresee the adoption of semi-supervised learning methods, to reduce the number of labeled sessions that are needed for traffic classification and, at the same time, to automatically detect and capture emerging behaviors that were originally not present in the labeled training set.

References

[1] Ericsson. (2018) Ericsson mobility report june 2018. [Online]. Available: https://www.ericsson.com/en/mobility-report/reports/june-2018

[2] Cisco. (2017) Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper. [Online]. Available: www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[3] S. Chen and J. Zhao, "The requirements, challenges, and technologies for 5g of terrestrial mobile telecommunication," *IEEE communications magazine*, vol. 52, no. 5, pp. 36–43, 2014.

[4] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning," in *Network Traffic Measurement and Analysis Conference (TMA)*. Vienna, Austria: IEEE, June 2018.

[5] C. Zhang, X. Ouyang, and P. Patras, "Zipnet-gan: Inferring fine-grained mobile traffic patterns via a generative adversarial neural network," in *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. Seoul-Incheon, South Korea: ACM, 2017.

[6] Z. Chen, B. Yu, Y. Zhang, J. Zhang, and J. Xu, "Automatic mobile application traffic identification by convolutional neural networks," in *IEEE Trustcom/BigDataSE/ISPA*. Tianjin, China: IEEE, August 2016.

[7] N. Bui and J. Widmer, "Owl: A reliable online watcher for lte control channel measurements," in *Workshop on All Things Cellular: Operations, Applications and Challenges*. New York, NY, USA: ACM, October 2016.

[8] EU EARTH: Energy Aware Radio and neTwork tecHnologies, "D2.3: Energy efficiency analysis of the reference systems, areas of improvements and target breakdown," Deliverable D2.3, www.ict-earth.eu, 2010.

[9] F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin, "Understanding Mobile Traffic Patterns of Large Scale Cellular Towers in Urban Environment," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, 2017.

[10] J. K. Laurila, D. Gatica-Perez, J. B. I. Aad, O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, and M. Miettinen, "The mobile data challenge: Big data for mobile computing research," in *Mobile Data Challenge Workshop (MDC), in conjunction with "Pervasive 2012"*, Newcastle, UK, 2012.

[11] G. Barlacchi, M. D. Nadai, R. Larcher, A. Casella, C. Chitic, G. Torrisi, F. Antonelli, A. Vespignani, A. Pentland, and B. Lepri, "A multi-source dataset of urban life in the city of Milan and the Province of Trentino," *Scientific Data*, vol. 2, no. 150055, pp. 1–15, 2015.

[12] *TSG RAN; NR; Overall description; Stage 2*, 3GPP TS 38.300, Release 15, v16.0.0, Jan. 2020.

[13] *TSG RAN; NR; Physical channels and modulation*, 3GPP TS 38.211, Release 16, v16.0.0, Dec. 2019.

[14] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: an open-source platform for LTE evolution and experimentation," in *ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization (WiNTECH)*. New York, NY, USA: ACM, October 2016.

[15] "E-UTRA; physical layer procedures," *3GPP TS, vol. 36.213*, 2016.

[16] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[17] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *International Conference on Machine Learning (ICML)*, Atlanta, USA, June 2013.

[18] T. Tieleman and G. Hinton, "Divide the gradient by a running average of its recent magnitude," *Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[19] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," in *International Conference on Artificial Neural Networks (ICANN)*. Edinburgh, UK: IEEE, 1999.

[21] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. The MIT Press, 2016.

[22] M. Gadaleta and M. Rossi, "IDNet: Smartphone-based gait recognition with convolutional neural networks," *Pattern Recognition*, vol. 74, pp. 25–37, 2018.

[23] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[24] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[25] Y. Wu and Y. Liu, "Robust truncated hinge loss support vector machines," *Journal of the American Statistical Association*, vol. 102, no. 479, pp. 974–983, 2007.

[26] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] C. E. Rasmussen, "Gaussian processes for machine learning," in *Advanced lectures on machine learning*. Springer, 2004, pp. 63–71.

[28] A. Furno, M. Fiore, R. Stanica, C. Ziemlicki, and Z. Smoreda, "A tale of ten cities: Characterizing signatures of mobile traffic in urban areas," *IEEE Transactions on Mobile Computing*, vol. 16, no. 10, pp. 2682–2696, 2017.

[29] D. Hendrycks, M. Mazeika, and T. G. Dietterich, "Deep anomaly detection with outlier exposure," in *International Conference on Learning Representations (ICLR)*, Vancouver, BC, Canada, April 2018.

[30] S. Sigurdsson, J. Larsen, L. K. Hansen, P. A. Philipsen, and H.-C. Wulf, "Outlier estimation and detection application to skin lesion classification," in *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*. Orlando, FL, USA: IEEE, May 2002.

[31] Y. Chen, X. Dang, H. Peng, and H. L. Bart Jr., "Outlier Detection with the Kernelized Spatial Depth Function," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 288–305, 2009.

[32] Y. Choi, J. Y. Chung, B. Park, and J. W.-K. Hong, "Automated classifier generation for application-level mobile traffic identification," in *IEEE Network Operations and Management Symposium (NOMS)*. Hawaii, USA: IEEE, April 2012.

[33] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service usage classification with encrypted internet traffic in mobile messaging apps," *IEEE Transactions on Mobile Computing*, vol. 15, no. 11, pp. 2851–2864, 2016.

[34] J. Yang, Z. Ma, C. Dong, and G. Cheng, "An empirical investigation into CDMA network traffic classification based on feature selection," in *International Symposium on Wireless Personal Multimedia Communications (WPMC)*. Taipei, Taiwan: IEEE, December 2012.

[35] X. Han, Y. Zhou, L. Huang, L. Han, J. Hu, and J. Shi, "Maximum entropy based IP-traffic classification in mobile communication networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*. Shanghai, China: IEEE, April 2012.

[36] R. Nix and J. Zhang, "Classification of android apps and malware using deep neural networks," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 1871–1878.

[37] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[38] ——, "Multi-classification approaches for classifying mobile app traffic," *Journal of Network and Computer Applications*, vol. 103, pp. 131–145, 2018.

[39] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE communications magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[40] P. Wang, X. Chen, F. Ye, and Z. Sun, "A survey of techniques for mobile service encrypted traffic classification using deep learning," *IEEE Access*, vol. 7, pp. 54 024–54 033, 2019.

[41] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[42] F. Meneghello, M. Rossi, and N. Bui, "Smartphone Identification via Passive Traffic Fingerprinting: a Sequence-to-Sequence Learning Approach," *IEEE Network*, vol. 34, no. 2, pp. 112–120, 2020.