# DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning

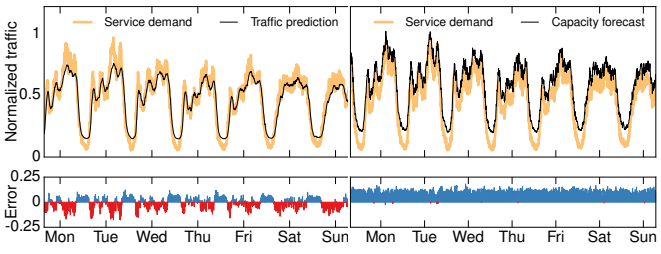
Dario Bega*†, Marco Gramaglia†, Marco Fiore‡ Albert Banchs*† and Xavier Costa-Perez§
*IMDEA Networks Institute, Madrid, Spain, Email: {dario.bega, albert.banchs}@imdea.org
†University Carlos III of Madrid, Madrid, Spain, Email: mgramagl@it.uc3m.es
‡CNR-IEIIT, Italy, Email: marco.fiore@ieiit.cnr.it
§NEC Laboratories Europe, Heidelberg, Germany, Email: xavier.costa@neclab.eu

***Abstract*—Network slicing is a new paradigm for future 5G networks where the network infrastructure is divided into slices devoted to different services and customized to their needs. With this paradigm, it is essential to allocate to each slice the needed resources, which requires the ability to forecast their respective demands. To this end, we present DeepCog, a novel data analytics tool for the cognitive management of resources in 5G systems. DeepCog forecasts the capacity needed to accommodate future traffic demands within individual network slices while accounting for the operator's desired balance between resource overprovisioning (*i.e.*, allocating resources exceeding the demand) and service request violations (*i.e.*, allocating less resources than required). To achieve its objective, DeepCog hinges on a deep learning architecture that is explicitly designed for capacity forecasting. Comparative evaluations with real-world measurement data prove that DeepCog's tight integration of machine learning into resource orchestration allows for substantial (50% or above) reduction of operating expenses with respect to resource allocation solutions based on state-of-the-art mobile traffic predictors. Moreover, we leverage DeepCog to carry out an extensive first analysis of the trade-off between capacity overdimensioning and unserviced demands in adaptive, sliced networks and in presence of real-world traffic.**

## I. Introduction

The next generation of mobile networks will enable an unprecedented heterogeneity of applications with very diverse Quality of Service (QoS) requirements [1]. *Network slicing*, allowing operators to customize Virtualized Network Functions (VNFs) for individual mobile services [2], will be key in accommodating such a diversified demand. However, the emergence of sliced networks also promises to skyrocket the complexity of resource management, moving from the rather limited reconfiguration possibilities offered by current Operations and Business Support System (OSS/BSS) to a rich, software-defined layer that manages thousands of slices belonging to hundreds of tenants on the same infrastructure [3].

**Network management and capacity forecast.** To cope with the new milieu, network operators are striving to make resource management and orchestration (MANO) processes highly automated. To realize the 5G principle of *cognitive network management* [4], two complementary technologies are needed: ($i$) technical solutions that enable end-to-end Network Function Virtualization (NFV), and provide the flexibility necessary for resource reallocation; and, ($ii$) data analytics that operate on mobile traffic measurement data, automatically identify demand patterns, and anticipate their future evolution.




(a) Legacy traffic prediction (b) DeepCog capacity forecast

Fig. 1: Top: actual and predicted weekly demands for Youtube at a datacenter controlling 470 4G eNodeBs. Bottom: levels of overprovisioning (blue) and capacity violations (red) over time. (a) Output of a recent deep learning predictor [7] of mobile traffic. (b) Output of DeepCog, tailored to anticipatory network resource allocation. Figure best viewed in colors.

From a technical standpoint, solutions that implement NFV at different network levels are well established, and start to be tested and deployed. Examples include current MANO platforms architectures like ETSI NFV [5], and implementations such as OSM [6], which allow to reconfigure and reassign resources to VNFs on the fly.

By contrast, the integration of data analytics in cognitive mobile networks is still at an early stage. Nowadays, resource assignment to VNFs is a reactive process, mostly based on hysteresis thresholding and aimed at self-healing or fault tolerance. There is a need for proactive, data-driven, automated solutions that enable cost-efficient network resource utilization, by anticipating future needs for capacity and timely reallocating resources just where and when they are required.

The focus of our work is precisely on the design of data analytics for the anticipatory allocation of resources in cognitive mobile networks. Specifically, we seek a machine learning solution that runs on traffic measurements and provides operators with information about the capacity needed to accommodate future demands at each network slice – a critical knowledge for data-driven resource orchestration.

**Related works on mobile traffic prediction.** Our problem is tightly linked to mobile traffic prediction, which is the object of a vast literature [8], [9]. Solutions to anticipate future offered loads in mobile networks have employed a variety of tools, from autoregressive models [10]–[12] to information theory [13], and from Markovian models [14] to machine

learning [7], [15]–[17]. However, we identify the following major limitations of current predictors when it comes to supporting resource orchestration in mobile networks.

($i$) Predictors of mobile traffic invariably focus on providing forecasts of the future demands that minimize some absolute error [8], [9]. This approach leads to predicted time series that deviate as little as possible from the actual traffic time series, as exemplified in Fig. 1a for a real-world case study. While reasonable for many applications, such an output is not sufficient for network resource orchestration. In this context, the operator aims at accommodating the offered load at *all* times, since resource underprovisioning implies high costs in terms of high subscribers' churn rates, as well as of significant fees for violating Service-Level Agreements (SLAs) with tenants. Yet, if an operator decided to allocate resources based on a legacy prediction like that in Fig. 1a, it would incur into capacity violations most of the time (as illustrated in the bottom subplot). Overdimensioning with respect to the forecast is just a rough fix, as traffic predictors do not provide insights on the required excess capacity.

We argue that a more effective anticipatory resource allocation can be achieved by designing machine learning solutions that anticipate the minimum provisioned capacity needed to cut down SLA violations. This would close the present gap between simple traffic prediction and practical orchestration: for instance, it would provide the operator with the explicit capacity forecast that mitigates underprovisioning in Fig. 1b.

In addition to the fundamental issue above, two other important shortfalls also affect state-of-the-art mobile traffic prediction, when applied to cognitive network management.

($ii$) With the adoption of network slicing, forecasts must occur at the slice level, *i.e.*, for specific mobile services in isolation. However, most traffic predictors are evaluated with demands aggregated over all services – an easier problem, since aggregate traffic yields smoother and more regular dynamics – and may not handle well the bursty, diversified traffic exhibited by each service.

($iii$) Existing machine learning predictors for mobile traffic typically operate at base station level [7], [16]. However, NFV operations mainly occur at datacenters controlling tens (*e.g.*, at the mobile edge) to thousands (*e.g.*, in the network core) of base stations. Here, prediction should be more efficient if performed on traffic at each datacenter, where orchestration decisions are taken, rather than combined from independent forecasts at base stations.

**Contribution.** In this paper, we present DeepCog, a new mobile traffic data analytics tool that is explicitly tailored to solve the capacity forecast problem exposed above. The design of DeepCog yields multiple novelties, summarized as follows:

- it hinges on a deep learning architecture inspired by recent advances in image processing, which exploits space-independent correlations typical of mobile traffic and computes outputs at a datacenter level;

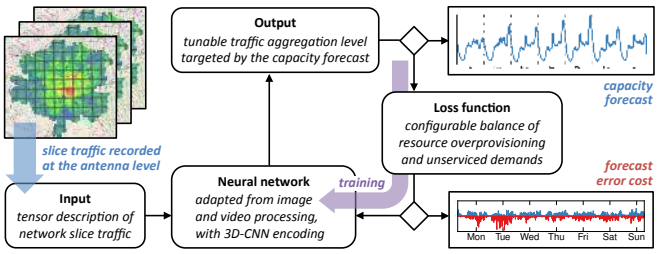



Fig. 2: Outline and interaction of the DeepCog components.

- it leverages a customized loss function that targets capacity forecast rather than plain mobile traffic prediction, and that lets the operator tune the balance between overprovisioning and demand violations;
- it operates on a per-service basis in accordance with network slicing requirements, and at datacenters located at different points of the mobile network.

Overall, these design principles jointly solve the different problems discussed before: in fact, Fig. 1b is an example of the required capacity forecast by DeepCog in a real-world case study. We remark that DeepCog is one of the very first examples of rigorous integration of machine learning into a cognitive network management process, and marks a difference from the common practice of embedding vanilla deep learning structures into network operation [9]. Extensive performance evaluations with substantial measurement data collected in an operational metropolitan-scale mobile network demonstrate the superiority of our approach over a wide range of benchmarks based on traditional and state-of-the-art mobile traffic predictors. In addition, DeepCog lets us investigate the fundamental trade-off between overprovisioning of resources and denied service demands, in practical case studies where network slicing and a sensible capacity forecast are in place.

## II. DeepCog Overview

The design of DeepCog is outlined in Fig. 2. Its organization is that typical of deep learning systems, and it stems from ($i$) properly formatted *input* data used to build the forecast, which is fed to ($ii$) a deep *neural network* architecture that extrapolates and processes input features to provide ($iii$) an *output* value, *i.e.*, the capacity forecast. During the training phase, the output is used to evaluate ($iv$) a *loss function* that quantifies the error with respect to the ground truth, and, in DeepCog, accounts for the costs associated to resource overprovisioning and service request denial. Below, we present each component, and discuss its mapping to the elements of a 5G network architecture with cognitive resource management.

**Input.** The input is composed by measurement data generated in a specific network slice, and recorded by dedicated probes deployed within the network infrastructure. Depending on the type and location of the probe, the nature of the measurement data may vary, describing the demands in terms of, *e.g.*, signal quality, occupied resource blocks, bytes of traffic, or computational load on VNFs. DeepCog leverages a set of transformations to map any type of slice traffic

measurements into a tensor format that can be processed by the learning algorithm. Details are provided in Section III-C.

**Neural network.** DeepCog leverages a deep neural network structure composed of suitably designed encoding and decoding phases, and that performs a next-step prediction. The structure is general enough that it can be trained to solve the capacity forecast problem for different network slices dedicated to services with significantly diverse demand patterns. The neural network structure is described in Section III-B.

**Output.** The learning algorithm returns a forecast of the capacity required to accommodate the future demands for services associated to a specific network slice. This generic definition of output can be specialized to distinct orchestration use cases, which typically differ by the traffic aggregation level at which the resource configuration takes place. DeepCog is designed for flexibility, and can serve heterogeneous orchestration scenarios. This is achieved by tailoring the very last layer of the deep neural network to the layout of datacenters at which the prediction must occur. Details are in Section III-B.

**Loss function.** Legacy deep learning solutions assess the quality of the output by means of standard loss functions, such as Mean Absolute Error (MAE) or Mean Square Error (MSE). However, these are not well suited metrics in the case of network capacity forecast. Here, prediction errors determine a certain (*e.g.*, economic) cost for the mobile network provider, whose nature depends on whether the capacity is overestimated or underestimated by the system. Overestimation leads to reserving unnecessary resources that will remain unused, hence reducing the efficiency. Underestimation means that not all of the network slice demand will be served, which reduces the QoS offered to the end users, and possibly incurs violations of SLAs with tenants providing the services in the network slice. DeepCog implements a novel loss function that captures the actual cost incurred by an operator in presence of errors in the capacity forecast. This allows training the learning algorithm so that it anticipates the amount of resources that achieve a minimum-cost balance of (high) end-user QoS and (low) overprovisioning. Our loss function can be configured to reflect a variety of economic cost strategies via a single parameter that has a clear interpretation. Details are in Section IV.

## III. Neural Network Architecture

In this paper, we deal with the capacity forecast problem, which lies in choosing the amount of resources allocated to a specific network slice in order to meet the demand for the services of that slice. This must occur within a target set of datacenter nodes, in a way that the economic cost incurred by the operator to accommodate the slice traffic is minimized.

### A. System model

In our network model, we consider that time is divided in slots, which we denote by $t$. Let $\delta_s^i(t)$ be the traffic associated with slice $s$ that is observed at base station $i \in \mathcal{N}$ and time $t$. A *snapshot* of the demand of slice $s$ at time $t$ is given by a set $\boldsymbol{\delta}_s(t) = \{\delta_s^1(t), \ldots, \delta_s^N(t)\}$, and provides a global view of the traffic for that slice at time $t$ across the whole network. We let

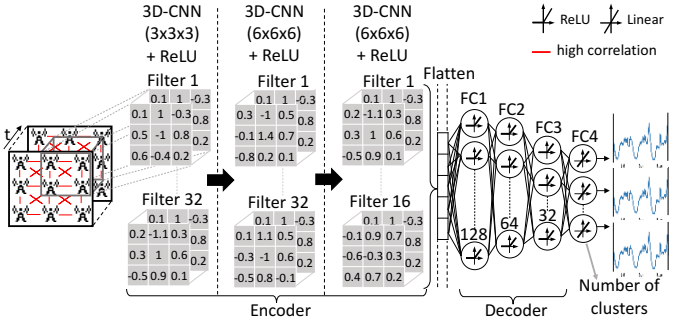



Fig. 3: DeepCog neural network encoder-decoder structure.

$\mathcal{N}$ denote the set of $N$ base stations in the network, and $\mathcal{M}$ the set of $M < N$ datacenters. Base stations are associated to datacenters via a surjective mapping $f : \mathcal{N} \to \mathcal{M}$, such that a datacenter $j \in \mathcal{M}$ serves the aggregated load of all of the associated bases stations, *i.e.*, $d_s^j(t) = \sum_{i|f(i)=j} \delta_s^i(t)$ for slice $s$ at time $t$. The set of demands across all datacenters is then given by $\mathbf{d}_s(t) = \{d_s^1(t), \ldots, d_s^M(t)\}$. Let us denote the capacity forecast for slice $s$ at datacenter $j$ and time $t$ as $c_s^j(t)$, and the set of capacities at all $j \in \mathcal{M}$ as $\mathbf{c}_s(t) = \{c_s^1(t), \ldots, c_s^M(t)\}$. Then, the capacity forecast problem is that of computing $\mathbf{c}_s(t)$ based on knowledge of the $T$ previous traffic snapshots $\boldsymbol{\delta}_s(t-1), \ldots, \boldsymbol{\delta}_s(t-T)$. The quality of the forecast $\mathbf{c}_s(t)$ with respect to the ground truth $\mathbf{d}_s(t)$ is measured by a loss function $\ell(\mathbf{c}_s(t), \mathbf{d}_s(t))$.

DeepCog solves the capacity forecast problem by means of a deep neural network architecture, and by accounting for a suitably designed loss function $\ell(\cdot)$. The design of the neural network entails: $(i)$ selecting and composing layers that efficiently solve the problem; $(ii)$ transforming the traffic snapshots $\boldsymbol{\delta}_s(t)$ for a specific network slice into a format that is consistent with that accepted by the first layer chosen for the neural network. Next, we separately discuss these aspects.

### B. Deep Neural Network Structure

The design of the neural network structure in DeepCog is inspired by recent breakthroughs [18] in deep learning for image processing. As summarized in Fig. 3, the network is composed of an encoder, which receives an input representing the mobile traffic data $\boldsymbol{\delta}_s(t-1), \ldots, \boldsymbol{\delta}_s(t-T)$, and maps important spatial and temporal patterns in such data onto a low-dimensional representation. Then, a decoder processes this rendering to generate the final capacity forecast $\mathbf{c}_s(t)$ at the desired set of datacenters $\mathcal{M}$. Below, we detail the encoder and decoder implementations, and discuss the training procedure.

**Encoder.** The encoder is composed by a stack of three three-dimensional Convolutional Neural Network (3D-CNN) layers [19]. Generic Convolutional Neural Networks (CNNs) are a specialized kind of deep learning structure that can infer local patterns in the feature space of a matrix input.

Since mobile network traffic exhibits correlated patterns in both space and time, we employ 3D-CNNs as the features to be learned are spatiotemporal in nature. Our choice is motivated by their excellent performance with fairly limited training: by exploiting the fact that our input data yields high

local correlation (as discussed in Section III-C) each neuron layer explores it through a limited *receptive field* (*i.e.*, a small portion of the input, fixed by the kernel size). These layers receive thus a tensor input $\mathcal{T}(\boldsymbol{\delta}_s(t-1)), \ldots, \mathcal{T}(\boldsymbol{\delta}_s(t-T))$, where $\mathcal{T}(\cdot)$ is a transformation of the argument snapshot into a matrix. Each neuron of the 3D-CNN layers runs a filter $\mathcal{H}\left(\sum_t \mathbf{I}(t) * \mathbf{K}(t) + \mathbf{b}\right)$ where $\mathbf{I}(t)$ is the input matrix for time $t$ (*i.e.*, $\mathcal{T}(\boldsymbol{\delta}_s(t))$ at the very first layer, for slice $s$, $*$ denotes the 3D convolution operator, $\mathbf{K}(t)$ is the kernel of filters, $\mathcal{H}(\cdot)$ is a non-linear activation function, and $\mathbf{b}$ is a bias vector.

The receptive field is set by $\mathbf{K}(t)$: as depicted in Fig. 3, there are two different kernel configurations. We used a $3\times3\times3$ kernel configuration for the first 3D-CNN layer, and a $6\times6\times6$ for the second and the third. Many different activation functions have been proposed in the literature, spanning from linear functions to $\tanh$, sigmoid or Rectified Linear Unit (ReLU). Among these, we select ReLU as $\mathcal{H}(\cdot)$, expressed as $\max(0, \mathbf{x})$, which is known to provide advantages in terms of discriminating performance and faster learning [20]. Finally, $\mathbf{b}$ is randomly set at the beginning of each training phase.

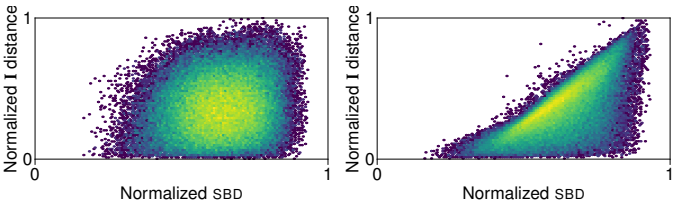
The second and third 3D-CNN layers are interleaved with Dropout layers: they regularize the neural network and reduce overfitting [20] by randomly set to zero a number of output features from the preceding layer during the training phase. The *dropout rate* defines the probability with which output features undergo this effect. During training, we employ two Dropout layers with dropout rate equal to 0.3.

**Decoder.** The decoder uses Multi-Layer Perceptrons (MLPs), a kind of fully-connected neural layers, *i.e.*, they interconnect every neuron of one layer with every neuron of the next layer. This provides the ability to solve complex function approximation problems. In particular, MLPs are able to learn global patterns in their input feature space [21]. In our structure, each layer performs an operation $\mathcal{H}'(\mathbf{x}\cdot\mathbf{W}+\mathbf{b})$, where $\mathbf{x}$ is the MLP layer input vector, $\mathbf{W}$ a weight matrix related to the neurons of each layer, and $\mathbf{b}$ the bias vector. $\mathbf{W}$ plays a similar role to $\mathbf{K}(t)$ in the encoder part: its values drive the prediction through the layers of the decoding part.

As for the activation functions $\mathcal{H}$, we employ ReLU for all MLP layers except the last one, where a linear activation function is used since the desired output takes real values. We highlight that the last linear layer can be configured to produce multiple predictions in parallel, each matching the aggregate capacity required by a subset of base stations. This is performed by training the network against a ground-truth $\mathbf{d}_s(t)$ that reflects the desired traffic aggregations. Ultimately, this lets us configure the DeepCog neural network to predict capacity at a datacenter level, for any configuration of $\mathcal{M}$.

**Training procedure.** We leverage the Adam optimizer, which is a Stochastic Gradient Descent (SGD) method that provides faster convergence compared to other techniques [22].

SGD trains the neural network model, evaluating at each iteration the loss function $\ell(\cdot)$ between the forecast and the ground truth, and tuning the model parameters in order to minimize $\ell(\cdot)$. We use the default Adam optimizer configuration, with a learning rate of $5\times10^{-4}$. We stress that, although a separate training is required for each network slice $s$, the neural network structure in Fig. 3 yields equally good performance with any service we could experiment with.




(a) Legacy tensor (b) DeepCog tensor

Fig. 4: Scatterplots of the actual correlation between traffic (x axis) and distance in the tensor input (y axis) for each pair of base station in our reference scenario. Left: legacy approach of [7], [9]. Right: novel approach adopted in DeepCog.

### C. Tensor Input

The 3D-CNN layer adopted as the first stage of the decoder requires a multidimensional tensor input. We thus need to define the transformation $\mathcal{T}(\cdot)$ of each traffic snapshot into a matrix. Note that 3D-CNN layers best perform in presence of a tensor input that features a high level of local correlation, so that neurons operate on similar values. In image processing, where close-by pixels typically have high correlation, this is easily solved by treating the pixel grid as a matrix. In line with this strategy, the current common practice in mobile network traffic prediction is to leverage the geographical locations of the base stations, and assign them to the matrix elements so that their spatial proximity is preserved as much as possible [7], [9]. However, this approach does not consider that correlations in mobile service demands at a base station level do not to depend on space, rather on land use [23]: base stations exhibiting strongly correlated network slice traffic may be far apart, *e.g.*, covering the different train stations within a same large city. Thus, we aim at creating a tensor input whose neighboring elements correspond to base stations with strongly correlated mobile service demands. We construct the mapping of base stations into a matrix structure as follows.

- We define, for each base station $i$, its historical time series of total traffic as $\boldsymbol{\tau}^i = \{\delta^i(1), \ldots, \delta^i(t-1)\}$, where $\delta^i(t) = \sum_s \delta^i_s(t)$. Then, for each pair $i$ and $j$, we determine the similarity of their recorded demands by computing $\text{SBD}^{ij} = f_{\text{SBD}}(\boldsymbol{\tau}^i, \boldsymbol{\tau}^j)$, where $f_{\text{SBD}}(\cdot)$ is the shape-based distance, a state-of-the-art similarity measure for time series [24]. All pairwise distances are then stored in a distance matrix $\mathbf{D} = (\text{SBD}^{ij}) \in \mathbb{R}^{M\times M}$.
- We compute virtual bidimensional coordinates $\mathbf{p}_i$ for each base station $i$ so that the values in the distance matrix $\mathbf{D}$ are respected as much as possible. Formally, this maps to an optimization problem whose objective is $\min_{x_1,\ldots,x_M} \sum_{i<j} (\|\mathbf{p}_i - \mathbf{p}_j\| - \text{SBD}^{ij})^2$, efficiently solved via Multi-Dimensional Scaling (MDS) [25].
- We match each point $\mathbf{p}_i$ to an element $e$ of the input matrix $\mathbf{I}$, again minimizing the total displacement. To this end, we: ($i$) quantize the virtual surface encompassing all points $\mathbf{p}_i$ so that it results into a regular grid of $N$ cells;

($ii$) assume that each cell is an element of the input matrix; ($iii$) compute the cost $k_{ie}$ of assigning a point $\mathbf{p}_i$ to element $e$ as the Euclidean distance between the point and the cell corresponding to $e$. We then formalize an assignment problem with objective $\min_a \sum_{i\in\mathcal{N}} \sum_{e\in\mathbf{I}} k_{ie}x_{ie}$, where $x_{ie} \in [0,1]$ is a decision variable that takes value 1 if point $\mathbf{p}_i$ is assigned to element $e$, and must fulfill $\sum_{i\in\mathcal{N}} x_{ie} = 1$ and $\sum_{e\in\mathbf{I}} x_{ie} = 1$. The problem is solved in polynomial time by the Hungarian algorithm [26].

The solution of the assignment problem is the transformation $\mathcal{T}(\cdot)$ of the original base stations into elements of the matrix $\mathbf{I}$. The mapping function $\mathcal{T}(\cdot)$ allows translating a traffic snapshot $\boldsymbol{\delta}_s(t)$ into matricial form, hence $\boldsymbol{\delta}_s(t-1), \ldots, \boldsymbol{\delta}_s(t-T)$ into the tensor required by the entry decoder layer in Fig. 3.

Fig. 4 provides an intuition of the improved representation granted by the DeepCog tensor input presented above. Each point in the scatterplots matches a pair of base stations $i$ and $j$ in the reference scenario that we use for our experiments (see Section V). The coordinates are $\text{SBD}^{ij}$, *i.e.*, the actual correlation between their traffic time series (x axis), and the Manhattan distance between the elements associated to $i$ and $j$ in $\mathbf{I}$ (y axis). The output of our approach is depicted in Fig. 4b; for such an approach, the measured Pearson's correlation coefficient $r^2$ is of 0.51. Instead, Fig. 4a depicts the output obtained from directly applying the assignment in the last step to the geographical locations of the base stations. Results shows that traffic similarity and position in $\mathbf{I}$ in this case are uncorrelated (the Pearson's correlation coefficient is of $r^2 = 0.02$), demonstrating that spatial proximity does indeed not imply traffic correlation. As a consequence, the latter approach is less suitable for a 3D-CNN.

As an important final remark, our proposed approach is general. The tensor input generation process presented before can be used with demands expressed in terms of, *e.g.*, signal quality, resource blocks, bytes, CPU cycles, or memory.

## IV. Loss Function

One of the key components of the proposed system is the *loss function*, denoted by $\ell(\cdot)$, which determines the penalty incurred when making a prediction error. We propose a novel loss function that is tailored to the specific requirements of the capacity forecast problem. Our design of $\ell(\cdot)$ accounts for the costs resulting from ($i$) forecasting a lower value than the actual offered load, which leads to an *SLA violation* due to the provisioning of insufficient resources, ($ii$) predicting a higher value than the actual one, which leads to *overprovisioning*, *i.e.*, allocating more resources than those needed to meet the demand. Then, $\ell(\cdot)$ must account for the penalty inflicted in each case to ensure that we drive the system towards an optimal trade-off between overprovisioning and SLA violations.

Recall that we denote by $c_s^j(t)$ the forecast for time $t$, *i.e.*, the provisioned capacity at datacenter $j \in \mathcal{M}$ and for network slice $s$, and by $d_s^j(t)$ the corresponding actual offered load. The cost incurred by the operator due to a discrepancy between $c_s^j(t)$ and $d_s^j(t)$ is quantified as follows.


(a) Ideal Model (b) Actual Implementation

Fig. 5: Cost model $\ell'(c_s^j(t) - d_s^j(t))$. Left: ideal model in (1). Right: actual implementation in (2).

- If the actual load is larger than the predicted one, *i.e.*, $c_s^j(t) < d_s^j(t)$, then we have an SLA violation for the target network slice. We assume that this yields a fixed cost $\beta$. Such cost may represent, for instance, the monetary compensation that the operator has to pay to a tenant whose SLA is not satisfied.
- If the actual load is smaller than the predicted one, *i.e.*, $c_s^j(t) > d_s^j(t)$, the operator has instead overprovisioned the network slice. If the (monetary) cost of one unit of capacity is $\gamma$, this yields a surcharge of $\gamma \cdot (c_s^j(t) - d_s^j(t))$.

If we define $x = c_s^j(t) - d_s^j(t)$, the above cost model can be expressed as follows:

$$\ell'(x) = \begin{cases} \beta & \text{if } x \leq 0 \\ \gamma \cdot x & \text{if } x > 0, \end{cases} \tag{1}$$

which is illustrated in Fig. 5a. Note that a perfect prediction that allows to exactly anticipate the required capacity, *i.e.*, $c_s^j(t) = d_s^j(t)$, maps to $x = 0$ in (1), and avoids any penalty.

As the loss function must steer capacity allocation to an optimal balance of the two costs above, the only factor that matters in its definition is the ratio between the costs of SLA violation and overprovisioning. Hence, a simpler equivalent expression is obtained by defining $\alpha \doteq \beta/\gamma$, and multiplying the two components by $1/\gamma$. The parameter $\alpha$ can be interpreted as the amount of overprovisioned capacity units that determine a penalty equivalent to one SLA violation: a larger $\alpha$ implies higher SLA violation fees for the operator.

However, the SGD method used to train the neural network does not work with constant or step functions, which forces us to introduce minimum slopes for $x < 0$ and at $x = 0$. The cost model implementation is:

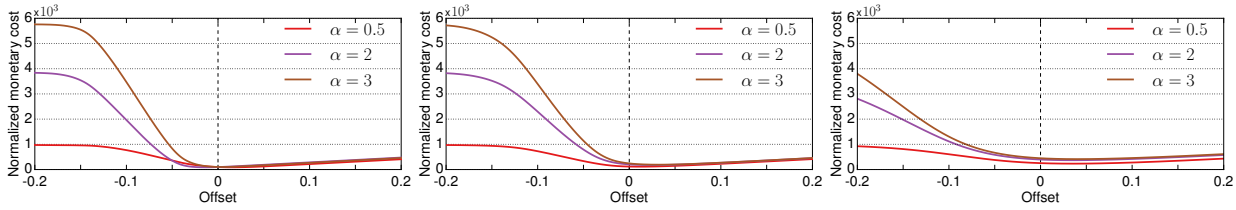
$$\ell'(x) = \begin{cases} \alpha - \epsilon \cdot x & \text{if } x \leq 0 \\ \alpha - \frac{1}{\epsilon}x & \text{if } 0 < x \leq \epsilon\alpha \\ x - \alpha\epsilon & \text{if } x > \epsilon\alpha, \end{cases} \tag{2}$$

where $\epsilon$ is a very low value that does not affect the shape of the cost, as per Fig. 5b, but allows SGD to operate correctly.

The loss function used to assess the quality of the solution to the capacity forecast problem at time $t$ for slice $s$ is then

$$\ell\left(\mathbf{c}_s(t), \mathbf{d}_s(t)\right) = \sum_{j\in\mathcal{M}} \ell'\left(c_s^j(t) - d_s^j(t)\right). \tag{3}$$

The cost model in (2), hence the loss function in (3), depend on a single parameter, $\alpha$. The setting of $\alpha$ can be obtained




(a) Facebook, core datacenter (b) Snapchat, MEC datacenter (c) Youtube, C-RAN datacenter

Fig. 6: Monetary cost (aggregated over time and normalized by the cost of one capacity unit) incurred when the overprovisioning level is shifted from that selected by DeepCog (at the abscissa origin). DeepCog accounts for unavoidable prediction errors, and balances overprovisioning and SLA violations so as to minimize the economic cost for the network operator. Each plot refers to one case study, *i.e.*, a combination of $(i)$ mobile service associated to a dedicated slice and $(ii)$ datacenter type.

TABLE I: Mobile services retained for dedicate network slices.

| *Service name* | *Service class* | *Traffic %* |
|---|---|---|
| YouTube | streaming | 27.3 |
| Netflix | streaming | 1.8 |
| Instagram | social media | 3.4 |
| Snapchat | messaging | 8.9 |
| Apple Store | online store | 10.5 |

| *Service name* | *Service class* | *Traffic %* |
|---|---|---|
| iTunes | streaming | 20.0 |
| Facebook | social media | 20.4 |
| Twitter | social media | 3.2 |
| Google Play | online store | 4.3 |
| Pokemon Go | mobile gaming | 0.1 |

from the monetary cost of violating an SLA (*i.e.*, $\beta$) and that of provisioning additional resources (*i.e.*, $\gamma$), and allows tuning DeepCog to any market strategy adopted by the operator.

Specifically, predicting future demands with complete accuracy is impossible due to unforeseeable fluctuations in the activity of the users. Then, in order to avoid frequent SLA violations, the operator needs to account for some level of overprovisioning. The level of overprovisioning should be chosen such that the resulting monetary cost is minimized. This is precisely what DeepCog does by hinging on (3). In order to show the quality of our solution, in Fig. 6 we run DeepCog in several representative case studies from our reference scenario (see Section V). For each case study, DeepCog advocates a level of overprovisioning (x-axis origin), which entails a given monetary cost (corresponding y value); we then vary the overprovisioned capacity by adding to or subtracting from the forecast capacity a fixed offset on (x axis) and observe how this affects the monetary cost (on the y axis). The curves confirm that DeepCog always identifies the overprovisioning level that minimizes the monetary cost for the operator, under inherently inaccurate prediction.

## V. Performance Evaluation

In order to evaluate DeepCog, we consider a real-world scenario, consisting of a mobile network deployed in a large metropolitan region of around 100 km$^2$. We leverage measurement data about the demands for individual mobile services in the target region, collected by the network operator and generated by several millions of users. In our experiments, we use demands expressed in bytes, and collected at the gateway of an operational mobile network by monitoring the GPRS Tunneling Protocol (GTP).

Independent network slices are then assigned to a representative set of services, listed in Tab. I. Our selection covers popular applications with diverse requirements in terms of bandwidth and latency: this lets us provide an adequate picture of the performance of DeepCog under the heterogeneous network traffic that characterizes current mobile service demands.

In order to assess the flexibility of DeepCog in serving heterogeneous NFV scenarios, we consider three different classes of datacenters where cognitive network management is run: $(i)$ a core network datacenter that controls all 470 4G eNodeBs in the target metropolitan area; $(ii)$ Mobile Edge Computing (MEC) datacenters that handle the traffic of around 70 eNodeBs each; $(iii)$ C-RAN datacenters located in proximity of the radio access, which perform baseband processing and scheduling for 11 eNodeBs each. The network is partitioned according to the methodology proposed in [27].

Capacity is predicted in terms of bytes of traffic that will have to be accommodated, which is a reasonable metric to capture for resource utilization in actual virtual network functions [28]. DeepCog can be configured to anticipate capacity over any time interval; in our experiments, we operate it on 5-minute time steps, by using the previous 30 minutes of traffic (*i.e.*, $T = 6$) arranged in a $47 \times 10$ matrix as an input[1]. Thus, the next-step forecast occurs over an horizon of 5 minutes. The rationale for this choice roots in practicality: resource reallocation updates in the order of minutes are typical for computational and memory resources in architectures implementing NFV [29], and are in line with those supported by any state-of-the-art Virtual Infrastructure Manager (VIM) [30].

We leverage the reference scenario described above to evaluate the performance of DeepCog. We employ two months of mobile traffic data for training and another two weeks of data for actual experiments. This setting is also used for all benchmark approaches. All results are derived with a high level of confidence and low standard deviation.

### A. Comparison with state-of-the-art traffic predictors

DeepCog is designed as a building block within a network resource orchestration framework. A fundamental advantage over existing solutions in the literature is that it targets capacity forecast, avoiding SLA violations, rather than a mere prediction of traffic load which may incur into frequent violations.

[1]We tested a number of different configurations of input snapshots, without significant differences in terms of results.

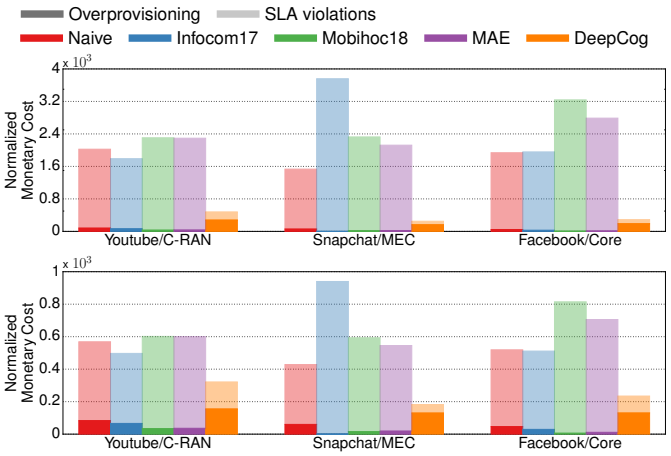



Fig. 7: Comparative evaluation of DeepCog with four benchmarks in three representative case studies. The monetary cost (normalized by the cost of one capacity unit) incurred by the operator is split into costs due to overprovisioning (dark) and SLA violations (light). Top: $\alpha = 2$. Bottom: $\alpha = 0.5$.

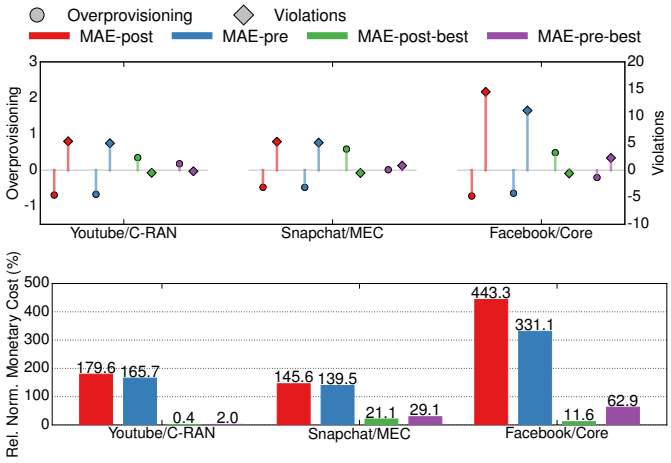



Fig. 8: Relative performance with respect to DeepCog of schemes that leverage legacy traffic prediction with additional overprovisioning offsets. Top: relative overprovisioning and SLA violations. Bottom: relative monetary cost.

In order to show this, we compare DeepCog against four benchmarks: ($i$) a `naive` technique that forecasts the future offered load by replicating the demand recorded at the same time during the previous week; ($ii$) the first approach proposed to predict mobile traffic based on a deep learning structure, referred to as `Infocom17` [7]; ($iii$) a recent solution for mobile network demand prediction that leverages a more complex deep neural network, referred to as `MobiHoc18` [16]; ($iv$) a reduced version of DeepCog, which uses the input, deep neural network structure and output described in Sections II-III, but replaces the loss function of Section IV with a legacy Mean Absolute Error (MAE) loss function[2], which results in a deep learning mobile traffic predictor, referred to as `MAE`.

The analysis considers three representative case studies for orchestration of resources: ($i$) a network slice dedicated to a video streaming service, *i.e.*, YouTube, at a C-RAN datacenter; ($ii$) a network slice reserved for a messaging service, *i.e.*, Snapchat, at a MEC datacenter; ($iii$) a network slice chartering traffic for a social network, *i.e.*, Facebook, at a core datacenter. Fig. 7 shows the results achieved by DeepCog and the four benchmarks above in these case studies. The plots show the normalized monetary cost for the operator, breaking down the cost due to allocating unnecessary resources (*i.e.*, overprovisioning) and to unserviced demands (*i.e.*, violations).

We observe that DeepCog yields substantially lower costs than all other solutions. The gain with respect to the *best* competitor for $\alpha = 2$ ranges between 273% and 381%, depending on the case study. The reason is shown in Fig. 1 at the beginning of this paper, which compares the output of `Infocom17` and DeepCog for one of the case studies. `Infocom17`, as all other benchmarks, targets mobile network traffic prediction, whereas DeepCog aims at forecasting capacity. As a result, DeepCog balances overprovisioning and SLA violations so as to minimize operation expenses; while `Infocom17` is oblivious to such practical resource management considerations. In other words, legacy predictors follow as closely as possible the general trend of the time series and allocate resources based on their output, which leads to systematic SLA violations that are not acceptable from a market viewpoint and incur huge fees for the operator. Instead, DeepCog selects an appropriate level of overprovisioning which minimizes monetary penalties (see Fig. 6). Indeed, even when choosing a low value such as $\alpha = 0.5$, which inflicts a small penalty for an SLA violation, DeepCog still provides gains up to 87% over the best performing benchmark by granting a suitable level of overprovisioning.

[2]We also experimented with other popular loss functions, *e.g.*, Mean Squared Error (MSE), with comparable results, omitted for space reasons.

### B. Comparison against predictors with overprovisioning

In the light of the above results, a more reasonable approach to resource allocation could be to consider a traditional mobile traffic prediction as a basis, and adding some *overprovisioning offset* on top of it. In order to explore the effectiveness of such an approach, we implement the following variants of `MAE`.

A first approach consists in adding an *a-posteriori* constant overprovisioning offset to the `MAE` output. This strategy, referred to as `MAE-post`, requires selecting a value of the static offset, which is then added to the predicted traffic. Based on the peak traffic activity observed in all historical data, we choose an offset 5%, which we deem a reasonable value. Alternatively, we also consider a best-case version of this solution, named `MAE-post-best`, where the a-posteriori overprovisioning is chosen by performing an exhaustive search over all possible offset values and selecting the one that minimizes the loss function $\ell(\cdot)$.

A second strategy is to account for some level of overprovisioning in a *preemptive* fashion, by introducing the offset during the deep neural network training. To this end, the

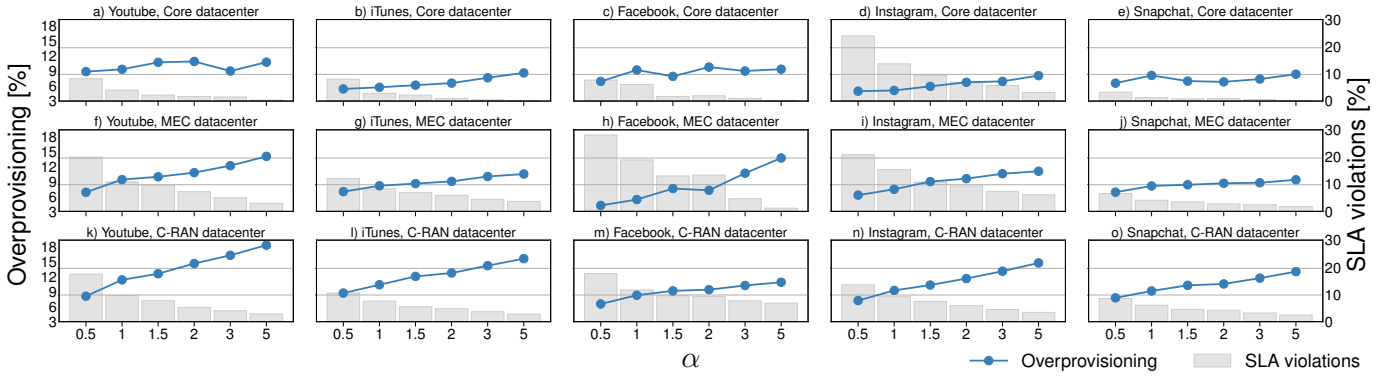



Fig. 9: Tradeoff between resource overprovisioning (expressed as a percentage of the actual demand) and SLA violation (expressed as a percentage of time slots), across 15 different scenarios, and in presence of $\alpha$ parameter.

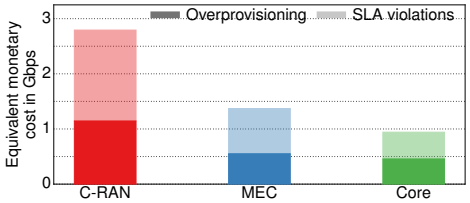



Fig. 10: Resource orchestration cost under network slicing and DeepCog capacity forecast, at three levels of the network.

`MAE-pre` solution replaces the MAE loss function with a new loss function $\mathcal{O} + \sum_{j \in \mathcal{M}} c_s^j(t) - d_s^j(t)$, where $\mathcal{O}$ denotes the a-priori overprovisioning offset. We set $\mathcal{O}$ equal to 5% of the peak traffic, which is a reasonable value also in this case. To compare against the best possible operation of this scheme, we also consider the `MAE-pre-best` technique, where $\mathcal{O}$ is set equal to the overprovisioning level selected by DeepCog.

We remark that the `MAE-post-best` and `MAE-pre-best` approaches are not feasible in practice, since they need to know the future (in an oracle-like fashion) in order to determine the best a-posteriori values for the offset and $\mathcal{O}$, respectively. Yet, they provide a benchmark for comparing the performance of DeepCog against the best possible solutions based on traditional mobile network traffic prediction.

Fig. 8 compares the performance of all the above solutions against DeepCog. The figure shows the oveprovisioned capacity, unserviced traffic, and total economic cost incurred by the operator in relative terms with respect to DeepCog. Results confirm that using a static overprovisioning in combination with mobile traffic prediction is largely suboptimal, both when the additional offset is considered preemptively or a-posteriori. Indeed, the two practical solutions considered, *i.e.*, `MAE-post` and `MAE-pre`, cause SLA violations that are two- to three-fold more frequent, resulting in an economic cost that is 140% to 400% higher than that granted by DeepCog.

Interestingly, even when parametrized with the best possible offsets, the approaches based on legacy traffic prediction cannot match the performance of DeepCog: `MAE-post-best` and `MAE-pre-best` dramatically reduce the penalties of their viable counterparts, yet lead to monetary costs that are up to 60% higher than those of DeepCog. We conclude that traffic predictors – no matter how they are enhanced – are not appropriate for the capacity forecast problem, for the simple reason that they are designed for a different purpose. Indeed, they ignore the economic penalties incurred by SLA violations, which drastically limits their ability to address this problem.

### C. Overprovisioning and SLA violation trade-off analysis

As discussed in Section II, DeepCog can accommodate capacity prediction at different traffic aggregation levels, for diverse slices, and under varied monetary cost strategies adopted by the operator. In the following, we capitalize upon this flexibility to carry out an extensive analysis of the fundamental trade-off between overprovisioning of resources and failing to meet service demands, in several practical scenarios. Specifically, our analysis considers five different network slices, dedicated to some popular services, *i.e.*, Youtube, iTunes, Facebook, Instagram, and Snapchat, and deployed at the three types of datacenters, *i.e.*, C-RAN, MEC, and network core.

Fig. 9 shows results for the above 15 scenarios under different economic strategies that are reflected by the $\alpha$ parameter of the loss function $\ell(\cdot)$, ranging from policies that prioritize minimizing overprovisioning over avoiding SLA violations ($\alpha = 0.5$) to others that strictly enforce the SLAs at the price of allocating additional resources ($\alpha = 5$). The plots show the two components that contribute to the total monetary cost: overprovisioning (given as a percentage of the actual demand) and SLA violations (given as a percentage of time slots). We observe that, as expected, higher $\alpha$ values reduce the number SLA violations (which become increasingly expensive) at the cost of provisioning additional capacity (which becomes cheaper). This tendency is consistent in all scenarios, which confirms that $\alpha$ effectively drives resource orchestration towards the desired operation point.

Our analysis also reveals that the level of overprovisioning grows as one moves from datacenters in the network core outwards. This is observed for all studied slices, and is due to the fact that more centralized datacenters work with increasingly aggregate traffic that is less noisy and easier to predict. Under such conditions, DeepCog needs a lower level of additional capacity to limit unserviced demands; indeed, the amount of SLA violations is typically lower at core datacenters.

### D. Orchestration results

We conclude our analysis by investigating the overall cost of resource orchestration in a sliced network. We consider an operational mobile network where an independent slice is dedicated to each of the services listed in Table I, and study three practical case studies, where DeepCog is used to drive the orchestration at C-RAN, MEC and core datacenters, respectively. We choose the operator policy as follows, based on the nature of each case study. SLA violations affect a large user population and shall be more expensive at the network core, where overprovisioning resources is cheaper; hence, we set a high $\alpha = 3$ in this case. Conversely, violations of SLAs at C-RAN datacenters concern a limited set of subscribers in a geographically constrained area, and are thus less costly; at the same time, deploying resources in proximity of the radio access is typically expensive; accordingly, we opt for $\alpha = 0.5$ in the C-RAN case study. Finally, we select an intermediate value, $\alpha = 1.5$, for the MEC case study. The results, given in Fig. 10, expose the monetary cost incurred by the operator in each case study. The cost is expressed in terms of Gbps and is a total over the reference metropolitan network scenario; it can easily translated into monetary units, based on the actual cost of provisioning one Gbps at each datacenter type. The cost values reflect that anticipatory resource orchestration is more efficient at the network core, and becomes increasingly complex as we move towards the edge: we quantify in 3:1 the ratio between the operational expenses at C-RAN with respect to the core.

## VI. Conclusions

In this paper we presented DeepCog, a novel data analytics tool for the cognitive management of resources in sliced 5G networks. Inspired by recent advances in deep learning for image and video processing, DeepCog leverages a deep neural network structure, which is trained using a customized loss function aiming at capacity forecast rather than legacy mobile traffic prediction. Ours is, to the best of our knowledge, the only work to date where a deep learning architecture is explicitly tailored to the problem of anticipatory resource orchestration in mobile networks. Thorough empirical evaluations with real-world metropolitan-scale data show the substantial advantages granted by DeepCog over state-of-the-art predictors, and provide a first analysis of resource orchestration costs at heterogeneous network slices and datacenters.

## Acknowledgments

The work of University Carlos III of Madrid was supported by the H2020 5G-MoNArch project (Grant Agreement No. 761445) and the work of NEC Laboratories Europe by the 5G-Transformer project (Grant Agreement No. 761536). The work of CNR-IEIIT was partially supported by the ANR CANCAN project (ANR-18-CE25-0011).

## References

[1] X. Foukas *et al.*, "Network Slicing in 5G: Survey and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, May 2017.

[2] P. Rost *et al.*, "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[3] A. de la Oliva *et al.*, "5G-TRANSFORMER: Slicing and Orchestrating Transport Networks for Industry Verticals," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 78–84, Aug. 2018.

[4] 5G-PPP, "The 5G Infrastructure Association. Pre-structuring Model, version 2.0," Nov. 2017.

[5] ETSI, NFVGS, "Network Function Virtualization (NFV) Management and Orchestration," *NFV-MAN*, vol. 1, Dec. 2014.

[6] ETSI, "OSM release FOUR technical overview," May 2018.

[7] J. Wang *et al.*, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *Proc. of IEEE INFOCOM*, May 2017, pp. 1–9.

[8] M. Joshi *et al.*, "A Review of Network Traffic Analysis and Prediction Techniques," *arXiv:1507.05722 [cs.NI]*, Jul. 2015.

[9] C. Zhang *et al.*, "Deep Learning in Mobile and Wireless Networking: A Survey," *arXiv:1803.04311 [cs.NI]*, Mar. 2018.

[10] F. Xu *et al.*, "Big Data Driven Mobile Traffic Understanding and Forecasting: A Time Series Approach," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 796–805, Sep. 2016.

[11] M. Zhang *et al.*, "Understanding Urban Dynamics From Massive Mobile Traffic Data," *IEEE Transactions on Big Data (to appear)*, 2017.

[12] S. T. Au *et al.*, "Automatic forecasting of double seasonal time series with applications on mobility network traffic prediction," *JSM Proceedings, Business and Economic Statistics Section*, Jul. 2011.

[13] R. Li *et al.*, "The prediction analysis of cellular radio access network traffic: From entropy theory to networking practice," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 234–240, Jun. 2014.

[14] M. Z. Shafiq *et al.*, "Characterizing and modeling internet traffic dynamics of cellular devices," in *Proc. of ACM SIGMETRICS*, Jun. 2011, pp. 265–276.

[15] A. Y. Nikravesh *et al.*, "An Experimental Investigation of Mobile Network Traffic Prediction Accuracy," *Services Transactions on Big Data*, vol. 3, no. 1, pp. 1–16, Jan. 2016.

[16] C. Zhang *et al.*, "Long-Term Mobile Traffic Forecasting Using Deep Spatio-Temporal Neural Networks," in *Proc. of ACM MobiHoc*, Jun. 2018, pp. 231–240.

[17] J. X. Salvat *et al.*, "Overbooking network slices through yield-driven end-to-end orchestration," in *Proc. of ACM CoNEXT*, Dec. 2018, pp. 353–365.

[18] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, Sep. 2014.

[19] Y. LeCun, "Generalization and network design strategies," *Connectionism in perspective*, pp. 143–155, Jun. 1989.

[20] G. E. Dahl *et al.*, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *Proc. of IEEE ICASSP*, May 2013, pp. 8609–8613.

[21] F. Chollet, *Deep learning with Python*. Manning Publications Co, 2018.

[22] D. P. Kingma *et al.*, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, Dec. 2014.

[23] A. Furno *et al.*, "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, Oct. 2017.

[24] J. Paparrizos *et al.*, "k-Shape: Efficient and Accurate Clustering of Time Series," in *Proc. of ACM SIGMOD*, Jun. 2015, pp. 1855–1870.

[25] I. Borg *et al.*, "Modern Multidimensional Scaling: Theory and Applications," *Journal of Educational Measurement*, vol. 40, no. 3, pp. 277–280, Sep. 2003.

[26] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, Mar. 1955.

[27] C. Marquez *et al.*, "How Should I Slice My Network? A Multi-Service Empirical Evaluation of Resource Sharing Efficiency," in *Proc. of ACM MobiCom*, Nov. 2018, pp. 191–206.

[28] J.-J. Kuo *et al.*, "Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture," in *Proc. of IEEE INFOCOM*, Atlanta, GA, USA, May 2017, pp. 1–9.

[29] V. Sciancalepore *et al.*, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in *Proc. of IEEE INFOCOM*, May 2017, pp. 1–9.

[30] J. Gil Herrera *et al.*, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sep. 2016.