Machine-Learning-Based 5G Network Function Scaling via Black- and White-Box KPIs

Raffaele Bolla
DITEN – University of Genoa
CNIT – S2N National Lab
Genoa, Italy
raffaele.bolla@unige.it

Roberto Bruschi
DITEN – University of Genoa
CNIT – S2N National Lab
Genoa, Italy
roberto.bruschi@unige.it

Franco Davoli
DITEN – University of Genoa
CNIT – S2N National Lab
Genoa, Italy
franco.davoli@unige.it

Chiara Lombardo
CNIT – S2N National Lab
Genoa, Italy
chiara@tnt-lab.unige.it

Jane Frances Pajo
Telenor Research
Fornebu, Norway
jane-frances.pajo@telenor.com

Beatrice Siccardi
DITEN – University of Genoa
Genoa, Italy
beatrice.siccardi@tnt-lab.unige.it

***Abstract*— The diffusion of the Fifth-Generation (5G) of mobile radio networks will be the main driver in the digital transformation towards a new hyper-connected society. In order to satisfy the stringent demands of 5G-ready applications over the limited resources available at the edge, scaling mechanisms become crucial to guarantee the performance levels envisaged for 5G. Such mechanisms must be able to automatically perform according to the real-time user demands, the availability of computing resources and the state of Network Functions (NFs) and applications. In this context, this paper proposes a deep learning model, based on Artificial Neural Networks (ANNs), for the dynamic and automated orchestration of NFs. The novelty of this model is its independence from specific 5G NF implementations; this is due to the nature of the Key Performance Indicators (KPIs) used in this work, which are related to both execution environment (standard "black-box" KPIs) and standard 5G APIs ("white-box" KPIs). Results obtained on the orchestration of a Session Management Function (SMF) reach an accuracy of 97~98% for the training and validation phases and above 95% for the deployed model, as well as higher overall accuracy by ~5% and computational resource savings with respect to a threshold-based scheme.**

Keywords—5G, Network Management, Machine Learning, SMF.

I. Introduction

The Fifth-Generation (5G) of mobile radio networks and edge computing technologies is expected to constitute the second wave of the Data Revolution [1] and to play a key role in the digital transformation towards a new hyper-connected society. In order to satisfy the stringent demand of 5G-ready applications, and to fully exploit the available network and computing resources, the distribution of application components and Network Functions (NFs) from the cloud to the edge of the network will become common practice.

Differently from cloud ones, edge computing facilities usually offer limited computing resources on highly heterogeneous hardware. In order to manage such resources as efficiently as possible, scaling mechanisms become crucial to guarantee the performance levels envisaged for 5G (and beyond) applications. However, the proper design of such mechanisms is definitely a non-trivial task: along with the already mentioned limitations of edge datacenters, the lifecycle management of network services and applications is performed on geographically distributed facilities, it must be fully automated and requires the injection and update of applications/services (or parts of them) at run-time, only when, where and for the time needed by 5G end-users and connected things. Moreover, the link between the Key Performance Indicators (KPIs) characterizing NFs/application components and CPUs is not straightforward, so that often one of the two categories is neglected. Nevertheless, their joint utilization would allow for a deeper understanding of the dynamics and efficiency of NFs/applications internal processes, and their dependency on the underlying hardware.

Nowadays, such mechanisms heavily rely on Artificial Intelligence (AI) and Machine Learning (ML) techniques for the autonomous, proactive, and cognitive management of the lifecycle of 5G applications and network functions. Most works in the state of the art on scaling decisions propose threshold-based solutions. Such thresholds can either be static (as in [2] and [3]) or dynamic (as in [4]). Static threshold solutions may lead to an oscillating pattern if the load frequently fluctuates around the thresholds. Moreover, in these solutions, few KPIs are analyzed to trigger the scaling decision. In particular, both [2] and [4] exploit only one metric, CPU utilization; in [3] CPU usage, RAM usage and Quality of Experience (QoE) are adopted. Among others, dynamic scaling for the automation of resource re-allocation is considered in [5], while the authors in [6] propose a system to dynamically scale stateful NFs by state disaggregation.

In this framework, this paper presents a Deep Learning (DL) model, exploiting Artificial Neural Networks (ANNs), for the dynamic and automated scaling of NFs and applications according to the Zero-touch network and Service Management (ZSM) paradigm [7]. The model performs the automated addition or removal of NF/application instances across the geographically distributed edge network, according to real-time demands, and is based on a multiclass classification able to combine metrics on both the NF instance and the computational resources hosting it. The use of ML in this context provides two benefits: the ability to easily consider the contribution of a large number of KPIs and that of achieving higher levels of accuracy, as it is not mandatory to divide the feature space in a linear fashion.

This model, based on recent specifications coming from the ETSI Experiential Networked Intelligence (ENI) standard [8], is not used merely for horizontal scaling, but it leverages on the slice-specific network and computing resource analytics provided by the Network Data Analytics Function (NWDAF) and the Management Data Analytics Function (MDAF) for the automated allocation of NF/application instances in the zones with higher traffic density. In this work, the model is applied to the 5G Session Management Function (SMF), which, being in charge of the establishment, modification and release of user sessions, can hugely benefit from the deployment of instances in the proximity of the User Equipment (UE).

The novelty of this work lays in the generated dataset, which comprises a large number (with respect to the previous works mentioned above) of data from two different sources: the execution environment (herein Kubernetes) and the 5G Service-based architecture (SBA). In further detail, the KPIs are a blend of "black-box" KPIs (e.g., CPU and memory utilization) and "white-box" KPIs (e.g., NF metrics, such as the number of Protocol Data Unit (PDU) sessions, etc.). Furthermore, since the white-box KPIs are extracted from 5G standardized interfaces (via the aforementioned analytics) that do not depend on neither the application nor the underlying hardware, the model proposed herein is both application- and implementation-independent.

Results obtained on the scaling of the SMF over a geographically distributed edge show that the DL model training and validation phases reach an accuracy of 97~98%, with the deployed model, also going above 95% for most numbers of available samples. Furthermore, the presented model is compared to a threshold-based one; results show a higher overall accuracy by ~5% and a significant saving of computational resources for the DL model.

The remainder of the paper is organized as follows. Section II introduces the DL model, while its implementation for service scaling is reported in Section III. The ML workflow is described in Section IV, with the performance of the model training and deployment in Sections V and VI, respectively. Finally, conclusions are drawn in Section VII.

II. Deep Learning Model

ANNs follow a bio-inspired approach by learning by example; as such, they have been gaining particular interest in modelling real-world systems, especially in the presence of noise and nonlinearity. There are two main approaches for ANN-based predictive modelling: (*i*) Classification, which predicts a category, and (*ii*) Regression, which predicts a quantity, both in a supervised fashion. The former approach is the one adopted in this work, while the latter, which allows introducing traffic forecasting to anticipate the traffic needs, is currently under consideration for an extension of this paper, with the goal of realizing a proactive scaling mechanism allowing for the reduction of the time between the need for scaling and the actual action. Both approaches aim at learning the function that best maps the input variables to the output variables (either as a category or as a quantity) through nonlinear approximations.

In this section, we outline the classification problem in the context of horizontal service scaling that will be used as basis for the ZSM solution's DL model. In more detail, a model is built based on a training dataset (i.e., historical data), which is generally organized into a table of N rows (i.e., samples) and $M+T$ columns (i.e., M features/input variables and T target/output variables); note that in the current example $T=1$. The number of features defines the number of neurons in the input layer. For a classification problem, the number of neurons in the output layer is defined by the number of classes/labels C (for instance, there are three classes in the sample problem – ScaleIN, Maintain and ScaleOUT – hence, $C=3$); for a regression problem, the number of neurons is determined by the number of targets (T). Between these two layers, there can be one or more hidden layer(s) with a variable number of neurons. In DL applications, the deeper the ANN becomes, the more complex problems it will be able to solve, but also with increased computational complexity. Regarding the anatomy of a neuron, the inputs are used to obtain a biased, weighted sum that is then fed to the activation function to produce the output. Some of the most commonly used activation functions include the rectified linear unit, sigmoid, softmax and hyperbolic tangent. Further details are provided in Section III. The optimal ANN structure (i.e., the number of hidden layers, neurons per layer and their corresponding activation functions) highly depends on the application and the dataset. This decision is usually based on experience, search strategies [9], Genetic Algorithms [10], k-fold cross validation [11], etc.

The model used in this paper is a feed-forward back-propagation ANN. It was chosen because of its straightforward implementation through Keras and Tensorflow and its simplicity. As a matter of fact, since the goal is classification, we do not need to use the more complicated feed-back ANNs (e.g., Recurrent NNs) which are more appropriate when doing regression. This simplicity should reflect on the training time which may be a sensitive variable for 5G applications with the most stringent requirements. Therefore, in future works, when comparing different ML models, great consideration will be devoted to the training time. The feed-forward back-propagation ANN is trained by iteratively updating the weights and biases to minimize the error between the predicted and expected target values in the dataset. Starting with a random initialization, predictions are calculated by feeding the data forward through the network. Then, computed according to a specific loss function (e.g., cross-entropy, mean absolute error, mean squared error, among others), the error is propagated backwards. The weights and biases are updated according to the chosen optimizer and learning rate. This forward-backward propagation process is repeated until the loss function converges or a specified maximum number of iterations is reached.

Open-source tools (such as scikit-learn [12], TensorFlow [13], Keras [14] and PyTorch [15]) are already widely available for building DL models in Python. In this work, recent versions of TensorFlow (which already has a tight integration with Keras) will be considered to jointly benefit from the high-level APIs of Keras and the low-level control of TensorFlow.

III. Service Scaling with Deep Learning

The proposed model exploits deep ANNs for the zero-touch orchestration of a NF/application instance and is meant to be introduced in ETSI ENI-compliant network platforms and fed by the metrics and analytics from the NWDAF and MDAF. The outcome is the automated addition or removal of the instances across the geographically distributed edge network according to the real-time demands. In particular, the scaling problem is modelled as a Multiclass Classification problem that seeks to categorize various high-/low-level metric combinations into "superstates" relating to scaling decisions.

Let $\boldsymbol{X}$ be the set of (potentially) relevant high-/low-level metrics, and Y be the scaling decision. The goal is to find the function f that best maps $\boldsymbol{X}$ to Y, based on a training dataset that consists of historical ($\boldsymbol{X}$, Y) combinations. In a multiclass classification problem, Y refers to the label that corresponds to the "superstates" that categorize the metric combinations into classes of scaling decisions, such as whether to **scale in**, **maintain** or **scale out** for a given sample of $\boldsymbol{X}$.

Given a training dataset, a ML classification algorithm seeks to divide the input space into decision regions corresponding to the C discrete classes, and builds a model by creating decision boundaries between these regions based on the statistical patterns in the dataset [16] [17]. It is important to note that the classes of a given dataset may not always be exactly separable; hence, the goal of the algorithm is to find the model f that optimizes the cost function (such as maximizing accuracy or minimizing loss, among others).

Suppose that the random variables $\boldsymbol{X}$ and Y evolve over time depending on the service demands, and each combined realization or sample $(\boldsymbol{x}^{(t)}, y^{(t)})$ in the training dataset is indexed in temporal succession, $t = 0, 1, \ldots$. In a probabilistic view of multiclass classification, the model f and its parameters are found by maximizing the likelihood function $\mathcal{L}(\{P(y^{(t)}|\boldsymbol{x}^{(t)}), t = 0, 1, \ldots\})$ or, equivalently, minimizing the loss function, $E = -\log(\mathcal{L})$ during the training phase, such that f best captures the statistical patterns between the samples [16] [9]; further details regarding the model we adopted can be found in Section V. Once a model has been built, it can then be used to automatically classify unlabeled samples of $\boldsymbol{X}$, for instance, with an independent test dataset or in an actual production environment. During these phases, the model computes the posterior probabilities for each class, $\{P(y = c|\boldsymbol{x}), c = 1, \ldots, C\}$, given the unlabeled input data, and the class corresponding to the highest $P(y = c^*|\boldsymbol{x})$ indicates the predicted label for each sample.

An ANN-based classifier is considered in this work to capture the statistical patterns between $\boldsymbol{X}$ and Y, in a supervised learning approach. This choice is mainly motivated by the ANN's ability in modelling real-world systems, especially in the presence of noise and nonlinearity. As described in [17], an ANN-based model is a "nested" mathematical function f_{ANN} such that for an L-layer network:

$$Y = f_{ANN}(\boldsymbol{X}) = f_L(\cdots f_2(f_1(\boldsymbol{X}))) \qquad (1)$$

where cases with $L > 2$ (or with more than one hidden layer between the input and output layers) are considered as deep ANNs. Generally, for layer $l = \{1, \ldots, L\}$,

$$y_l = f_l(\boldsymbol{z}) = g_l(\boldsymbol{W}_l \boldsymbol{z} + \boldsymbol{b}_l) \qquad (2)$$

where $\boldsymbol{z}$ is the output of the preceding layer, g_l is the activation function, $\mathbf{W}_l$ is the weight matrix and $\mathbf{b}_l$ is the bias vector. The rows in the matrix $\mathbf{W}_l$ are vectors $\{\boldsymbol{w}_{l,m_l}\ ,\ m_l = 1, \ldots, M_l\}$, each with the same dimensionality as $\boldsymbol{z}$, where M_l is the number of neurons in layer l. It can be noted how the output layer generates a probability distribution $\{P(y = c|\boldsymbol{x}), c = 1, \ldots, C\}$, based on which the predicted label $\hat{y}$ can be obtained as the one corresponding to the highest $P(y = c^*|\boldsymbol{x})$ value.

Most of the model parameters shown above are not solely learned from the training samples, but also from the outputs of the preceding layers, especially in Deep Learning applications [17]. In this view, a ***feed-forward backpropagation*** ANN starts with a random initialization of the model parameters and calculates the predictions (e.g., with a Maximum Likelihood criterion) by feeding the data forward through the network; it then computes the error according to a specific loss function (e.g., *cross-entropy* or *negative log likelihood*) and propagates it backwards to update the weights and biases. As pointed out in [16], the learning process involves solving a nonlinear optimization problem and evaluating the derivatives of the loss function with respect to the model parameters; backpropagation is a technique for efficiently evaluating the gradient of the loss function for training feed-forward networks.

The most popular choices of activation functions in recent works are the rectified linear unit (ReLU) and the Softmax functions. ReLU is typically used in the hidden layers, while the activation function used in the output layer highly depends on the application – for instance, for a multiclass classification problem, Softmax is typically used [17] [9]. As previously anticipated, training an ANN entails solving an optimization problem – that is, minimizing the loss function. This can be performed through a variety of optimization algorithms, or the so-called "optimizers". In this work, we use the Adaptive Moment Estimation (Adam) [18].

IV. Machine Learning Workflow Applied to the 5G Environment

This section describes how the service scaling model introduced in Section III is applied to the 5G context, along with the setup that was used for the training and testing trials reported in Sections V and VI. ML workflows usually start with ***Data Collection*** from one or more data sources, carefully selected according to their potential relevance to the problem at hand. Since the acquired (raw) data may have different formats, especially if they come from several sources, the raw data need to undergo ***Pre-processing***, in which they are cleaned and formatted into a usable dataset, suitably structured for the problem; feature analysis, label/target definition and splitting of the dataset (into training and test sets) are also done in this phase. Once the structured dataset is ready, then comes the ***Modelling*** phase – where the training set is fed to the algorithm to learn the model parameters; in some cases, a validation set can be used to further refine (e.g., fine-tuning of parameters) the model. In the ***Deployment*** phase, the resulting model is tested with an independent test set, and/or in a production environment.

A. *Data Collection*

In this work, we consider the presence of a Prometheus database [19] which maintains the time series that the NWDAF and the MDAF can obtain from the SMF on the connection and mobility management tasks and on the NF usage statistics, respectively. The testbench used to generate the dataset and validate the model is shown in Fig. 1. In particular, for mimicking the presence of a 5G network, from the access (including the data generated by the users) to the core, we use UERANSIM [20] and Free5GC [21], respectively. The former is a Linux-based emulator of a complete 5G access network, including UEs and gNodeBs, while the latter provides a complete 5G core with the individual NFs provided in a containerized fashion, including our targeted SMF. Both tools are open source; UERANSIM is deployed through multiple Virtual Machines (VMs) and Free5GC relies on Kubernetes [22]. In each test, the number of UEs is randomly selected between 0 and 100 and gNodeBs vary between 1 and 3, resulting in varying system workloads. Furthermore, the 5G core network is container-based and runs on top of a Kubernetes cluster. Each SMF instance has resources limited to one virtual CPU (vCPU) and 512 MB of random-access memory (RAM). This configuration has been chosen to reach the maximum capacity of an SMF in the tests, and hence, the system will need to scale out or add another SMF instance.

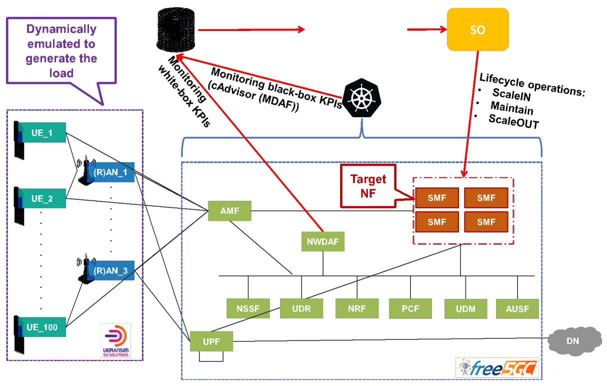

Fig. 1. Testbench used to generate the dataset.

The tests have been performed in an offline environment, such that each test has a corresponding Prometheus export which contains both the white-box and the black-box KPIs. In order to retrieve the former and passing them to Prometheus, since the current support of the REST APIs for events subscription in Free5GC NFs is incomplete, we extended the NWDAF to access the SMF logs (via the *Loki* framework). The latter are exported from Kubernetes to the Prometheus database through *cAdvisor* [23] which provides data regarding resource usage and performance characteristics on a container basis. The KPIs are exported to Prometheus every 1 second. The Prometheus data generated for the aforementioned tests are parsed to obtain the necessary features and data samples, then aggregated according to the desired structure. As previously mentioned, in order to generate a usable dataset, the data need to be carefully aggregated with consistent timeframes, and then, suitably structured for the problem at hand.

B. *Pre-Processing*

By looking at the (*created*, *responded*, *closed*) timestamps exported from Prometheus, 8 white-box KPIs can be derived per second: number of initiated and released Protocol Data Unit (PDU) sessions, number of allocated QoS Flow Identifiers (QFIs), user plane path changes, N10 and N11 terminations, tracking areas and Non-Access Stratum (NAS) interactions. From the data exported by *cAdvisor*, 6 black-box KPIs from the CPU utilization (*user*, *system*, *total*) and from the memory utilization (*rss*, *working_set*, *total*) are considered. The black-box metrics are further processed to obtain the average, min and max values per second across all active SMF instances, which results in a total of 32 features.

Data cleaning usually involves handling missing and/or invariant data. For instance, in the case of missing data, the distribution of the missing values can be initially checked; if there are only few missing values, one can either drop the samples or use the average value of the samples; otherwise, drop the columns with many missing values. After cleaning, the number of input features to be used in the actual training/testing is

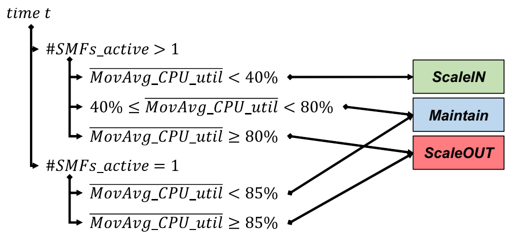

Fig. 2. Example of service scaling conditions.

reduced from 32 to 24 features. Given the structured (and cleaned) dataset, it can now be split into training and test sets; the former can also be further split for the actual training and the intermediate validation. In this work, the training set comprises 60% of the samples (20% of which was reserved for the validation in the model fitting), and the remaining 40% was used in the final testing.

Let us recall that the three classes – ***ScaleIN***, ***Maintain*** and ***ScaleOUT*** – act as "superstates" classifying the metric combinations towards the decision of whether to scale in, maintain or scale out the SMF instances. The input features in the dataset characterize the historical metric combinations, while the corresponding labels characterize the scaling decisions, where the latter is usually based on technical expertise.

Since the initial dataset is not labelled, we suppose that the historical scaling decisions are governed by some conditions based on the CPU utilization and the number of active instances, and generate the labels accordingly. However, it is important to note that in real-world ML/AI applications such conditions are supposed to be unknown to the developer, and learned by the model, on the basis of the given features and labels of the dataset. The sole purpose of the following conditions is to generate a working dataset.

In more detail, 1-minute moving averages of the CPU utilization (=*%total*) of each SMF instance are obtained and the mean across all active instances is considered:

$$\overline{MovAvg\_CPU\_util}^{(t)} = \frac{\sum_{\tau=t-T-1}^{t}\sum_{v=1}^{V^{(\tau)}}\left(\%total_v^{(\tau)}\right)}{T\times V^{(t)}} \qquad (3)$$

where $T = 60$ (seconds) and $V^{(t)}$ is the value of *#SMFs_active* at time instant t. Based on the runtime values of $\overline{MovAvg\_CPU\_util}^{(t)}$, the supposed conditions are shown in Fig. 2.

To better illustrate the service scaling scenario, Fig. 3 shows how $\overline{MovAvg\_CPU\_util}$ and *#SMFs_active* vary with time, as well as the desired scaling decision at each instant. It can also be observed that the notifications for the need of scaling in/out SMF instances continue until the service orchestrator actuates the desired decision.

Finally, it is worth noticing that the dataset we generated is purely synthetic. Nevertheless, we reiterate once again that our contribution lays in the composition

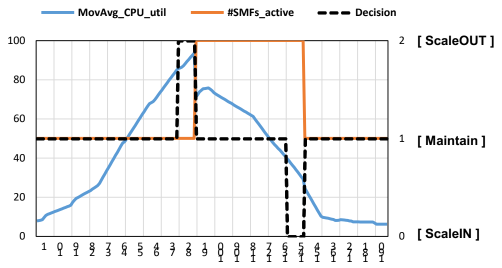

Fig. 3. Example of service scaling decisions based on CPU utilization and number of active instances.

of said dataset and not its real values. In the future we hope to test the model presented herein on real data.

V. Model Training and Validation

In this work, the widely used TensorFlow-Keras bundle is adopted for building the ANN-based model. With Keras, ANNs can be constructed in a modular and fully configurable way using its built-in classes for various components (such as the layers, loss functions, activation functions and optimizers, among others). It also features automatic backpropagation that simplifies the training process for the users.

A *Sequential* model [24] is considered for the multi-class classification problem at hand, and it is built by basically stacking ANN layers, each with exactly one input and output tensor (i.e., multi-dimensional array). *Dense* layers [25] are used in the model – particularly, 2 fully-connected hidden layers with 24 neurons/layer (i.e., set to equal the number of input features, *dataset.shape*[1]–1) and ReLU activation functions, then 1 fully-connected output layer with 3 neurons (i.e., set to equal the number of classes) and *SoftMax* activation functions. *Adam* is chosen as optimizer, with a learning rate η of 0.01, and default exponential decay rates (i.e., β_1 = 0.9 and β_2 = 0.999) [18]. Lastly, *sparse categorical crossentropy* and *sparse categorical accuracy* are used as loss function and metric, respectively, since the labels are not one-hot encoded and kept their integer values {0, 1, 2} (i.e., as on the right in Fig. 3).

Now, recalling the anatomy of the neuron, with a dense layer (fully connected), the number of inputs received by each neuron is equal to the number of neurons in the preceding layer, and its output is therefore also received by all neurons in the succeeding layer. Moreover, each neuron has one bias parameter. With this in mind, each hidden layer in the model constructed above has a total of 600 (=24 * 24 + 24) trainable parameters, while the output layer has 75 (=3 * 24 + 3). All in all, the model has 1275 trainable parameters. The training set used as input to the model has 4129 (60% of the total) samples – 80% of which are used for actual training, while the remaining 20% is used for validation. Fig. 4 shows the training and validation accuracies obtained in 100 epochs. It can be observed that the

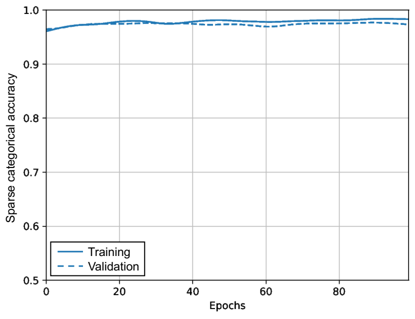

Fig. 4. Training and validation results.

model has stable accuracies at around 97~98%. It is also interesting to note that with 3 hidden layers, such accuracies start to decrease with increasing epochs, owing to overfitting. Finally, it is worth mentioning that the model was trained on a VM with 4 vCPUs and 8 GB of RAM; with this configuration the training phase lasted 30.48 seconds.

VI. Model Testing and Deployment

This phase comprised the testing of two models: a threshold-based model and the ANN model thoroughly explained in this text. The former is a static threshold model that is based on the one presented in [3]. In particular, thresholds are applied to both the black-box and the white-box KPIs; eventually the result is decided by an OR operator between the two sets. In both cases, 2753 samples (40% of the total) were used in this phase; these samples were *never-before-seen* samples, in order to reflect an actual production environment. One way of showing the performance of a classification model is through a *confusion matrix* [17], which summarizes the success of predicting the classes of the test samples, with one axis indicating the true label and the other the predicted label. Figs. 5 and 6 show the confusion matrices obtained with the two aforementioned models. In both cases we can observe the confusion matrices in terms of both the number of samples (Figs. 5(a) and 6(a)) and the percentages (Figs. 5(b) and 6(b)). By comparing the results shown in Figs. 5(b) and 6(b), it can be observed that the ANN-based classifier has a higher accuracy than the threshold-based one; in particular, the former was able to correctly classify {99.62%, 95.29%, 97.06%} of the samples belonging to the classes {0, 1, 2}, respectively.

It can also be noted in Figs. 5(a) and 6(a) that the sample set belonging to class 2 is substantially small (~10%) compared to the other two classes; hence, the dataset is class imbalanced. This can provide indications on which performance metrics to look at.

Starting from the confusion matrix, different performance metrics can be derived – namely, the ***Overall Accuracy***, ***Precision***, ***Recall*** and ***F1-score***. The Overall Accuracy is given by the ratio of the number of correctly classified samples to the total number of samples.

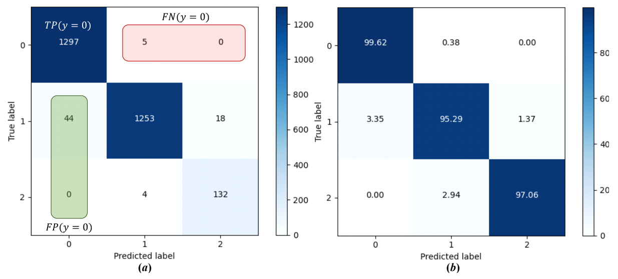

Fig. 5. Confusion matrix of the ANN-based model in terms of: (a) the number of samples, and (b) percentage.

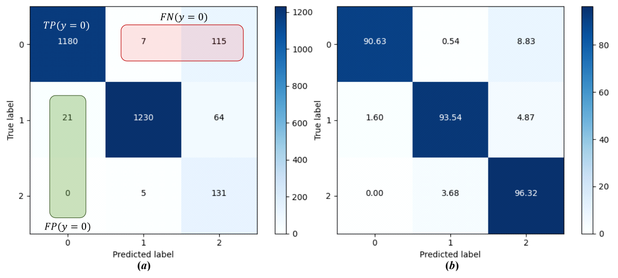

Fig. 6. Confusion matrix of the threshold-based model in terms of: (a) the number of samples, and (b) percentage.

$$Overall\ Accuracy = \frac{\sum_c TP(y=c)}{\#\ of\ test\ samples} \tag{4}$$

While this is a great metric for symmetric datasets, it is not the proper measure for class imbalanced datasets. In such cases, the other three metrics are usually adopted. Precision is given by the ratio between the number of *true positive* (TP) predictions and the total number of positive predictions (i.e., TP) plus the *false positive* (FP) predictions.

$$Precision(y=c) = \frac{TP(y=c)}{TP(y=c)+FP(y=c)} \tag{5}$$

This is a great metric for cases when the cost of FPs is high, such as in service scaling where FPs resulting in the *over-provisioning* of resources generate higher operational costs [9]. Recall is given by the ratio between TPs and the total number of positive test samples (i.e., TP plus the false negative (FN) predictions) [16].

$$Recall(y=c) = \frac{TP(y=c)}{TP(y=c)+FN(y=c)} \tag{6}$$

In contrast to Precision, this is a good metric for cases when the cost of FNs is high, such as in service scaling where FNs resulting in the *under-provisioning* of resources generate (potentially severe) degradations to the quality of service (QoS). Finally, the F1-score is the weighted average of the last two performance metrics, which is given by

$$F1\text{-}score(y=c) = \frac{2 \times Precision(y=c) \times Recall(y=c)}{Precision(y=c) + Recall(y=c)} \tag{7}$$

TABLE I. CLASSIFICATION REPORT OF THE ANN MODEL

Class	*Precision*	*Recall*	*F1–score*	*Samples*
0 (ScaleIN)	0.967	0.996	0.981	1302
1 (Maintain)	0.992	0.953	0.972	1315
2 (ScaleOUT)	0.880	0.971	0.923	136
Overall accuracy: 0.974				2753

Figs. 5(a) and 6(a) indicate the samples giving the number of TPs, FPs and FNs for class 0.

Table I shows the classification report of the ANN-based model for the independent test set, indicating the values for the Overall Accuracy and the per-class Precision, Recall and F1-score. It can be observed that the values of all the performance metrics are high; class 2 obtained the lowest performance (though still at around 88%~97%) due to the small number of samples available in both training and testing. Finally, it is worth mentioning that the overall accuracy of the threshold-based model is equal to 0.923; therefore, the ANN model's one is ~5% higher.

Finally, recalling again Figs. 5(a) and 6(a), the threshold-based classifier has a significant lower classification accuracy (by ~9%) in the 0 class, which corresponds to the scaleIN action. Therefore, by using the ANN-based classifier, one can avoid over-provisioning, which consists of the allocation of unneeded resources. This can be further observed in Fig. 7, which shows the plots of the allocated vCPUs in time for 254 test samples taken randomly from a uniform distribution among the test set. In Fig. 7, three plots are shown; they represent respectively the allocated vCPUs for the ideal, the threshold-based and the ANN-based case. Regarding the DL model, it can be observed that initially the allocated vCPUs are higher than in the other two cases; this is due to the resources devoted to the ANN classifier, which are added to those needed by the SMF instances. On the contrary, in the threshold-based model the added resources to classify the samples were not considered, since it requires a negligible computational effort. The actuation time is not considered since it is the same in both models (ANN- and threshold-based). Furthermore, in the ANN-based plot the delay due to the sample classification is considered; however, it cannot be noticed from the figure, since the delay is equal to 0.132 milliseconds, which is a lot smaller compared to the interval between samples (1 second). The ideal plot represents the number of allocated vCPUs in the case in which the classification resources are negligible and in which the samples are always classified correctly. Comparing the three plots, it can be observed that the ANN-based model allows to save a significant amount of resources (20 vCPUs maximum in this example) compared to the threshold-based one and that it closely follows the ideal plot except for some classification errors.

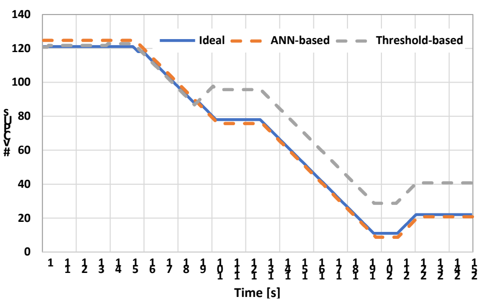

Fig. 7 Number of allocated vCPUs in time for 254 test samples taken randomly from the test set for three cases.

VII. CONCLUSIONS AND FUTURE WORK

This paper has proposed a DL model, which exploits deep ANNs, to drive the dynamic and automated scaling of 5G applications and network services. Our main contribution is the novel composition of the dataset. Thanks to the usage of both standard black-box (execution environment) and white-box (standard 5G APIs) KPIs, the proposed model is independent of specific 5G NF implementations. Results have been obtained by applying the proposed model on an SMF and have shown that the training and validation phases reach an accuracy of 97~98%, with the deployed model going above 95% for most numbers of available samples, proving higher overall accuracy by ~5%, and computational resource savings with respect to a threshold-based one.

In future works we plan to, on the one hand, compare this model to other ML models (e.g., other types of ANNs) and, on the other hand, use a real dataset.

Moreover, this work can be further extended by turning it into a proactive scaling mechanism. This can be achieved by the introduction of traffic forecasting (e.g., regression) techniques, through which it would be able to anticipate the traffic needs. The benefit of a proactive scaling mechanism lies in the reduction of the time between the need for scaling and the actual action. Another possible extension of this work consists of adding a new black-box KPI: the power consumption of the machine hosting the NF. This addition would contribute to fostering greener 5G technologies.

ACKNOWLEDGMENT

This work has been supported by the Horizon 2020 5G-PPP Innovation Action 5G-INDUCE (Grant Agreement no. 101016941).

REFERENCES

[1] S. Mian, "Investing in the Coming Data Revolution," NASDAQ News, Nov. 2019, https://www.nasdaq.com/articles/investing-in-the-coming-data-revolution-2019-11-06.

[2] G. A. Carella, M. Pauls, L. Grebe and T. Magedanz, "An Extensible Autoscaling Engine (AE) for Software-based Network Functions," 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2016, Palo Alto, CA, Nov. 2016, pp. 219-225, doi: 10.1109/NFV-SDN.2016.7919501.

[3] S. Dutta, T. Taleb and A. Ksentini, "QoE-aware Elasticity Support in Cloud-Native 5G Systems," 2016 IEEE International Conference on

Communications (ICC), Kuala Lumpur, Malaysia, May 2016, pp. 1-6, doi: 10.1109/ICC.2016.7511377.

[4] A. Beloglazov and R. Buyya, "Adaptive Threshold-Based Approach for Energy-Efficient Consolidation of Virtual Machines in Cloud Data Centers," Proc. 8th International Workshop on Middleware for Grids, Clouds and e-Science 2010 (MGC '10), Bangalore, India, Nov. 2010. Association for Computing Machinery, New York, NY, USA, Article 4, 1–6. https://doi.org/10.1145/1890799.1890803.

[5] W. Rankothge, E. Ramalhinho and J. Lobo, "On the Scaling of Virtualized Network Functions," Proc. 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, April 2019.

[6] F. B. Carvalho, R. A. Ferreira, Í. Cunha, M. A. M. Vieira and M. K. Ramanathan, "Dyssect: Dynamic Scaling of Stateful Network Functions," Proc. IEEE INFOCOM 2022, London, UK, May 2022.

[7] Zero touch network and Service Management (ZSM), ETSI ISG, https://www.etsi.org/technologies/zero-touch-network-servicemanagement.

[8] ETSI Experiential Networked Intelligence (ENI) home page at https://www.etsi.org/committee/eni?tmpl=component.

[9] T. Subramanya and R. Riggio, "Machine Learning-Driven Scaling and Placement of Virtual Network Functions at the Network Edges," in Proc. 2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, June 2019, pp. 414-422.

[10] G. Kousiouris et al., "Parametric Design and Performance Analysis of a Decoupled Service-Oriented Prediction Framework Based on Embedded Numerical Software," IEEE Transactions on Services Computing, vol. 6, no. 4, pp. 511-524, Oct.-Dec. 2013.

[11] X. Zhang, T. Xue and H. Eugene Stanley, "Comparison of Econometric Models and Artificial Neural Networks Algorithms for the Prediction of Baltic Dry Index," IEEE Access, vol. 7, pp. 1647-1657, 2019.

[12] "Scikit-learn." URL: https://scikit-learn.org/.

[13] "TensorFlow." URL: https://www.tensorflow.org/.

[14] "Keras." URL: https://keras.io/.

[15] "PyTorch." URL: https://pytorch.org/.

[16] C. M. Bishop, "Pattern Recognition and Machine Learning," Springer Science+Business Media, LLC, New York, NY, 2006.

[17] A. Burkov, "The Hundred-Page Machine Learning Book," Andriy Burkov, 2019.

[18] "Optimizer that implements the Adam algorithm," URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam.

[19] "Prometheus", URL: https://prometheus.io/.

[20] "UERANSIM", URL: https://github.com/aligungr/UERANSIM

[21] "Free5GC", URL: https://www.free5gc.org/

[22] "Kubernetes", URL: https://kubernetes.io/.

[23] "cAdvisor", URL: https://github.com/google/cadvisor.

[24] "The Sequential model," URL: https://www.tensorflow.org/guide/keras/sequential_model.

[25] "Densely-connected NN layers," URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense.