Utilizing Deep Learning for Mobile Telecommunications Network Management

Bálint Gyires-Tóth, Pál Varga
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics
2 Magyar Tudósok krt., Budapest, Hungary, H-1117
Email: {toth.b,pvarga}@tmit.bme.hu

Tamás Tóthfalusi
AITIA International Inc.
Telecommunication Division
Budapest, Hungary
Email: tothfalusi@aitia.ai

***Abstract*—Traditional network and service management methods were based on counters, data records, and derived key indicators, whereas decision were mostly made in a rule-based manner. Advanced techniques have not yet got into the everyday life of operators, mostly due to complexity and scalability issues. Recent progress in computing architectures, however, allows to re-visit some of these techniques – nowadays, especially artificial neural networks –, and apply those in the network management. There are two aims of this paper. First, to briefly describe the possible network and service management tasks within the mobile core, where utilizing deep learning techniques can be beneficial. Secondly, to provide a living example of predicting certain fault-types based on historical events, and to project further, similar examples to be executed on the same architecture. The real-life example presented here aims to predict the occurrence of certain errors during the VoLTE (Voice over LTE) call establishment procedure. The methods and techniques presented in this paper has been validated through live network monitoring data.**

I. Introduction

NETWORK and service management is such an applied research domain, where new methods and techniques get quickly utilized when some instrumentation becomes available. There are some basic questions keep popping up that have not yet been completely answered and solved – this may explain the origin of the word: re-search. The original question in this domain would be "how to manage networks and services in an optimal way"? The requirements include continuous and error-free availability, flexibility to satisfy the users needs, prompt and precise accounting, minimal resource usage (i.e. operating efficiency), and secure, fraud-free operation. These requirements later got structured by FCAPS functions: Fault, Configuration, Accounting, Performance, and Security Management, respectively. Issues raised within these areas have been tackled by the methods and technologies available at the given time – e.g., post-mortem with screwdrivers (then), or predictively with Artificial Neural Networks, ANNs (nowadays).

Deep Learning (DL) generally utilizes ANNs with many hidden layers, complex connections and novel algorithms, thus, these architectures can effectively learn representations. Furthermore, representation learning is jointly performed with model training, which often makes the approach more efficient, than other Machine Learning (ML) methods trained with extracted – and not learned – features. The instrumentation for their effective application has become available in recent years, in the form of high-performance GPUs. The other main ingredient for DL application is the vast amount of data to "learn" from. This traditionally gets generated during network operations, although, the practical access to such data is another issue, even after the advent of Big Data.

The motivation of this paper is to suggest DL methods as a possible solution for some re-appearing issues of the network and service management domain. The paper describes a practical, real-life example for this, in the area of mobile telco' network management. Our use-case is predicting future error occurrence based on statistical features of preceding message processing delays (also referred to as time differences).

The supervised model is trained on data gathered from a nationwide network. The training dataset consists of messages with their time stamps and nodes IDs. The input features are statistical features of *message processing times* at certain nodes and the outputs are the *number of occurring errors*. The training set was captured by a live network monitoring system; and the data for validation and testing are also provided by the same system, but at a different time-window.

II. Application Areas of Deep Learning in the Management of the Mobile Core

A. Related Work

Application suggestions for ML as well as DL techniques in the telecoms field may be categorized into many clusters. A comprehensive survey on ML for networking is provided by [1], whereas the authors of [2] present a more specific survey on mobile and wireless networking. While both touch upon network management and specifically on fault and security management, they both miss the mapping with the FCAPS model [3], which provides a complete coverage of the field in theory. The authors of [4] structure some of the ML techniques in the view of FCAPS, hence using the same approach as our current paper. These, very recent surveys list another dozens of earlier surveys in the area, making it somewhat superfluous to overview the domain in here. Still, it is worth categorizing actual live implementations at the telecommunication domain, since there are a few application types come to sight.

978-3-903176-15-7

The first application type is generic, with business-level view and more focusing of having "Big Data" [5] than using specific algorithms (such as ANNs). The second type of applications are targeting network (resource) optimization – such as optimal distribution of base stations, network nodes, etc. –, with various ML techniques [6]. The third type of applications target security areas, mostly zero day attacks with anomaly detection [7], [8], and other frauds [9]. The fourth type of application is related to traffic classification [1], for further usage in improving QoE through resource optimization, or for the identification of fraudulent traffic. While these applications have broad representation within the industrial forums, various other network and service management tasks that can also gain from utilizing ANNs, remain less visible.

B. TMN FCAPS functions

While it is a relatively old standard, the so called "logical model" of the Telecommunications Management Network (TMN) [3] is not at all obsolete. The elements of the FCAPS model – summarized by Figure 1 – is able to serve as a generic guideline aiming for covering various management areas of current networked services.

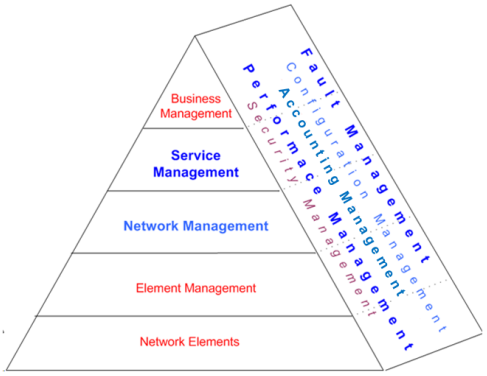

Fig. 1. The TMN FCAPS logical model [3]

Table I suggests further applications of DL in the network and service management levels, structured by the FCAPS model. Unlike described by [4], our novel table presented here suggests prediction, anomaly detection and clustering / classification tasks, rather than learning techniques.

III. Application Example: Predicting errors based on message transfer delays

The application example described in this section serves network management purposes for the Voice over LTE service. Call establishment procedures often end up with various reject codes that may or may not stem from actual network errors. It takes human expertise to build a knowledge base on the categorization of these cases related to VoLTE [10]. We have used this experience to apply supervised learning in the presented example. The data used in this study was captured at a nationwide mobile network operator.

For a comprehensive description of the monitoring solution of the LTE Evolved Packet Core and the IP Multimedia Subsystem (IMS) – that provided the data for our analysis –, please refer to our previous works [10], [11].

A. Data sources

According to a VoLTE call-setup procedure in the IMS domain, the initial control message of a Session Initiation Protocol (SIP) session is the INVITE method [12]. Generally, this message is the largest in size, because it contains the Session Description Protocol (SDP) as an embedded part. The SDP part of the first initial-INVITE contains the most information about the session preparation phase, since the caller includes all capabilities about the user data properties.

The initial-INVITE message requires outstanding processing at the IMS node side (e.g., the Session Call Session Control Function, S-CSCF) to locate the callee's IMS domain and bind resources to the new session). Beside the resource allocation and next-hop location, the given node often must parse the SIP content or even the embedded XML or SDP protocol content, as well. Altogether the processing time could get even 60-80 ms at specific nodes (e.g., Application Server, S-CSCF), which is a significant network delay during a call-setup procedure. To sum up the delays per node, it can often reach the magnitude of several hundreds of milliseconds. That is the reason why SIP protocol defines provisional response messages to inform the nodes that the call-setup is under processing.

Due to lack of resources, the control messages can suffer from congestion, which may increase the processing time and start a domino effect. The increased network delay may result packet retransmission events (i.e., further load), or may result error responses to a call setup request. One of the management targets here can be to predict such cases and intervene when (or before) the first dominoes start to fall.

Since SIP INVITE method is the most crucial in the call-setup sequence, we merely focused on it in our methodology.

The typical monitoring points for tracking SIP protocol messages are within the IMS domain: the ingress and egress ports of the Call-Session Control Function and Application Servers. In order to have a comprehensive analysis, other IMS entry points should be also monitored (e.g. Multi-service Access Node - MSAN, Session Border Control - SBC, or other gateway and border control functions of the operator).

Since the IMS domain has a distributed architecture and the nodes often come from different manufacturers, some of the SIP message fields are often changed during the processing and forwarding phase. To follow the complex call-setup message sequence hop-by-hop, we need extra information about the field contents' transformation. In our previous work [11] we investigated various use-cases and their key parameters for cross-correlation.

Based on the distributed mobile telecommunications network *monitoring solution* at a nationwide operator, we captured a 24 hour sample of SIP traffic and analyzed it in detail. Because the current system was not designed for ML purposes, data exportation is very slow. Thus, the current work is intended to be a proof of concept for future developments.

TABLE I
DEEP LEARNING APPLICATION EXAMPLES OF FCAPS TASKS IN THE MOBILE TELECOMMUNICATIONS DOMAIN

Management area	Prediction	Anomaly Detection	Clustering or Classification
Fault management	Fault Prediction; Automated Mitigation	Fault Detection; Root Cause Analysis	Alarm Correlation
Configuration management	Resource Optimization (SDN, Base Station power adjustment, Cloud resource allocation)	Configuration pattern recognition	Realizing similarities or differences in node configs
Accounting management	Churn prediction; Service utilization prediction	Misuse or Fraud	Traffic characterization; Usage profiling
Performance management	Utilization prediction (feeding Config. mgmt.)	Detecting under- or over-utilization of segments	Resource Planning QoS and QoE correlation
Security management	Intrusion Prevention	Detection of suspicious activities DDoS Detection	Intrusion Detection

During this time interval we captured about 66 million INVITE messages. The captured file contained key parameters about INVITE methods between different IP addresses.

Using the cross-correlation parameters (Call-ID and IMS-Charging-ID) from our previous work [11], we generated an own ID, namely Call-Summary-ID (CSID), for each INVITE message. The CSID is used to follow the control traffic hop-by-hop. In order to have clearly defined records to feed the DL algorithms with, the next step was the creation of a comma separated value (csv) file, with the following columns:

1) Timestamp,
2) Source IP,
3) Destination IP,
4) CSID,
5) Status code of the response message to the INVITE.

The status code of the response message is used to differentiate the error events. We consider the 4xx, 5xx and 6xx status code classes as an error response, except the following codes:

- 401 Unauthorized,
- 407 Proxy Authentication Required,
- 486 Busy Here,
- 487 Request Terminated.

Codes 401 and 407 are often (re-)authentication requests from the IMS. The 486 response means that the callee could not accept the call, and the 487 status code ends a cancelled session. This expert knowledge on practical categorization of SIP reject codes have been further detailed in [10].

B. Data preparation

The first step was to sort the CSV file based on the CSID and the timestamp information. As a result, we got INVITE message groups, in which the INVITE methods belong to the same call-setup procedure, ordered by time.

After gathering the data of one day from the described sources, the *features* consisted of a timestamp, a session identifier, status codes and two IP addresses, that are two nodes in the signaling. There were altogether 64,538,731 entries and 38,051,225 sessions in the data. Sessions with only one entry were dropped. There were 27,065,354 such sessions, and after dropping them 37,473,377 entries and 10,985,871 sessions remained. The time difference between each communication in every session was calculated, separately. As time difference cannot be computed in the case of first elements of the sessions, these were dropped, resulting in 26,487,506 entries.

To provide a view of the delays in this complex and distributed network architecture, we also grouped the IP addresses into five node groups. Without such a node type grouping the neural network would have been forced to cope with several dozens of "different" nodes. Beside the calculations taking superfluously long, such an approach would have suffered from lack of data volume at many node-pairs, and noisy data at many other pairs.

We used our own and the operators' engineering expert knowledge to group the node types for this given purpose. Our node clustering method is based on the trusted and untrusted domain definition from the operator's viewpoint; the typical entry points into the IMS domain and the architectural positions. There were 1935 unique nodes with different IP addresses, that resulted in 51 different node types based on the IP address range. Then the node types were grouped into the following five node categories:

1) Node0: other operator's node
2) Node1: untrusted domain entry points (e.g.: MSAN, IBCF - Interconnection Border Control Function)
3) Node2: trusted domain entry points (e.g.: SBC, PCSCF)
4) Node3: ICSCF, SCSCF nodes
5) Node4: Application servers

Message processing delays were then analyzed as Figure 2 suggests. This categorization may not be perfect, but helps decreasing the problem space without introducing major misunderstandings to the method.

Thus, IP pairs formed ten node category pairs: Node1 & Node1, Node1 & Node2, Node1 & Node3, Node1 & Node4, Node2 & Node2, Node2 & Node3, Node2 & Node0, Node3 & Node3, Node3 & Node4, Node0 & Node0.[1] From the status

[1]According to Figure 2 Node1 & Node3, Node1 & Node4, Node2 & Node0 and Node0 & Node0 category pairs should not exist. However, in real-world networks, signalling between such nodes also occurs.

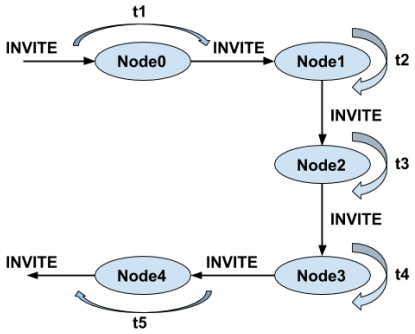

Fig. 2. Simplified view of INVITE messages getting delayed at Nodes

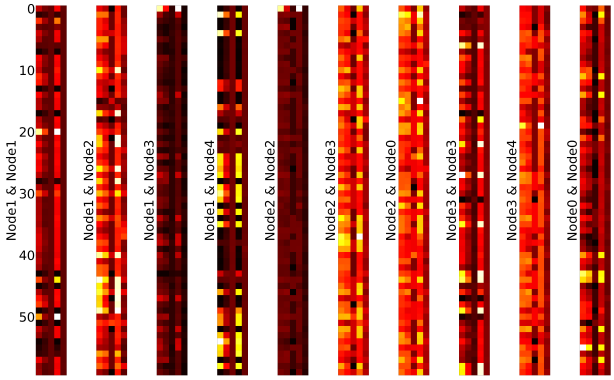

Fig. 3. A sample input with 60 time steps, 5 features (mean, standard deviation, minimum, maximum and median of time differences) and 10 category pairs. Warmer colors denote higher, darker colors denote lower values.

codes, as discussed in subsection III-A, the error occurrences were defined for each entry.

As a further step, the data was resampled, and the mean, standard deviation, minimum, maximum and median of time differences (called: *features*), as well as the sum of the errors (called: *output*) were aggregated for each of the node category pairs for every time step[2]. After examining the data, it typically showed a different distribution than Gaussian. Therefore were the minimum, maximum and median values included besides mean and standard deviation, as [13] suggests. The resulting data were used as the basis of inputs and outputs of the deep neural network.

C. Methodology

The goal was to predict the number of errors in the next time step for every node category pairs, based on the features of the current and preceeding time steps. Consequently, the *inputs* of the DL model are the calculated statistical *features* of the current and preceeding time steps, whereas the *outputs* are the number of errors in the next time step.

To model the temporal nature of the data, we applied two-dimensional convolutional neural networks [14]. The network was fed with three-dimensional input, derived from the pre-processed data, with the following dimensions: (*time steps, features, category pairs*). The corresponding output included the number of errors in the next time step for the ten category pairs. An example of the input with 60 time steps, five features, and ten category pairs is shown in Figure 3. Inspecting the image, in this 60 time steps interval Node2 & Node3 and Node2 & Node0 had the highest (as these parts are 'warmer') and Node1 & Node3 and Node2 & Node 2 had the lowest time differences (which is denoted by darker colors). Please also note that Node1 & Node4 had some significant changes is time differences (warmer colors followed by darker colors, and vice versa).

This visual representation is similar to a "weather report" over a map, and we can see darker areas – cyclone storms – moving over designated areas as time goes by.

The applied neural network is shown in Figure 4. As we did not suppose correlation among the features, but the time steps and the category pairs, in the first convolutional layer after the input a kernel with full-width (that was 5 in this case) was applied. In subsequent layers, the kernel width was set to 1.

To catch multi-scale temporal correlations, three different branches were used, with kernel height of 3, 5 and 7. In this case the height refers to consecutive time steps, so, 3, 5 and 7 time steps were considered in the three branches, respectively. In convolutional layers, stride was set to 1. The number of filters was 8 in the first two, and 16 in the second two convolutional layers. To help multi-scale event modeling, skip connections from inner parts of the convolutional blocks were introduced. Two maxpooling were also applied in each branch. Thus, the output dimension of the three branches were 12x1x16, 9x1x16 and 6x1x16. The dense layers had 128 neurons each, and the output dense layer had 10 neurons, one for the number of errors in each category pair. To avoid overfitting, dropout with 50% probability [15] in various parts of the network, and L2 weight regularization with 0.0001 ratio to trainable layers were added. The number of trainable parameters of the neural network was cca. 300k.

For training, the ADAM stochastic optimization method [16] was used, that utilizes adaptive per-parameter learning rate based on the first and second moments of the gradient. The initial learning rate was set to 0.001. After the validation loss was not decreasing for 100 epochs, training was stopped and the model with the best validation loss was loaded. To find a robust model architecture and near optimal hyperparameters, random search and manual tuning was performed.

IV. Evaluation and Results

The data preparation step was performed with one and ten seconds resampling. In both cases a sliding window of 60 was used with step size 1. Thus, the input involved 60 and 600 seconds of previous, the output 1 and 10 seconds of subsequent data. The number of training samples in the resulting database was 86361 and 8637, respectively.

Some of the features (mean and standard deviation) along the whole dataset for the category pairs are shown by Figure 5.

[2]The data is resampled in time steps. E.g., resampling at 1 second results in 60 seconds $\times$ 60 minutes $\times$ 24 hours $=$ 86400 time steps in one day data.

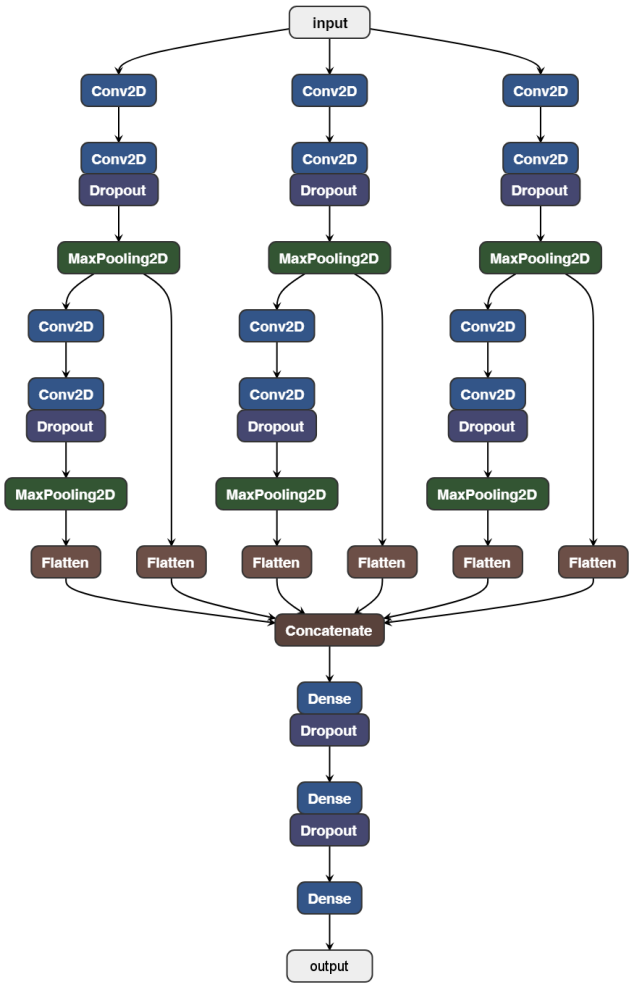

Fig. 4. The applied deep convolutional neural architecture.

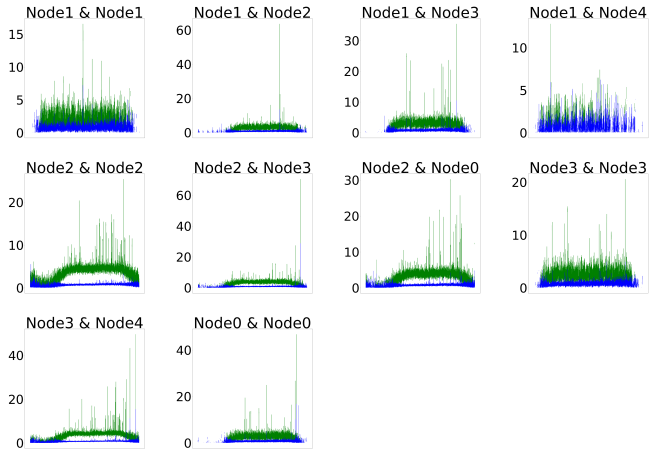

Fig. 5. Two input features of the ten node category pairs (1 second resampling). Y axis (please note, that it has different scales): Mean (blue) and standard deviation (green) of time differences. X axis: time, duration of one day (from 0:00 to 23:59).

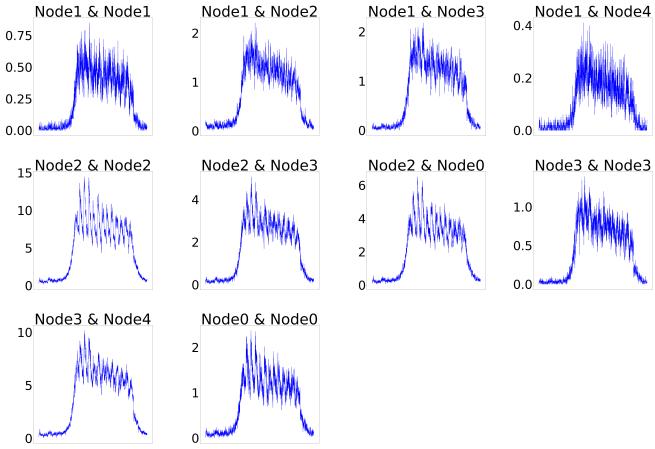

Fig. 6. Outputs (number of errors in the next time step) for the node category pairs (1 second resampling). Y axis (please note, it has different scales): moving average of the number of errors in the next time step (window size: 100). X axis: time, duration of one day (from 0:00 to 23:59).

The corresponding *outputs* – the number of errors in the next time step for all the category pairs within the given day – are shown in Figure 6.

The output of all the node category pairs show similar distribution, however, the number of errors differs in all node categories. Node2 & Node2 had the highest, while Node1 & Node4 had the lowest number of errors in average. The number of errors is low during the night, it peaks in the morning, it slowly decays until the late afternoon, and in the evening it decreases rapidly to low values again. Inspecting the mean and standard deviation (Figure 5) some nodes category pairs show similar characteristics (e.g., Node2 & Node2, Node2 & Node0, Node3 & Node 4), while others has fast transients (e.g., Node1 & Node 1, Node 1 & Node 4, Node 3 & Node 3).

The 50% of the data was used for training, 20% for validation and 30% for testing. To preserve the temporal nature, the splitting was done on the chronologically ordered data.

The network learned the problem in both cases with mean squared errors measured on test data shown in Table II. As the table shows, in case of 10 seconds resampling the mean squared error showed a similar pattern as in case of 1 second resampling, however, the errors were higher, as the target values had higher value range, indeed, over longer period.

The predictions and ground truth of the test database are shown in Figure 7. The test set represents the last 30% of the data chronologically, so it refers to cca. from 17h to 24h of the day. This part of the data was unseen by the model. The figure shows that the convolutional neural network successfully learned to predict the number of errors in the next time step based on the statistical features of previous 60 time step. In some cases, e.g. Node2 & Node2 and Node2 & Node0 faster transients were successfully modeled: the red line partially follows the first three peaks of the blue line (denoted by blue circles). In other cases, e.g., Node3 & Node4, it couldn't catch these variations (denoted by red circles).

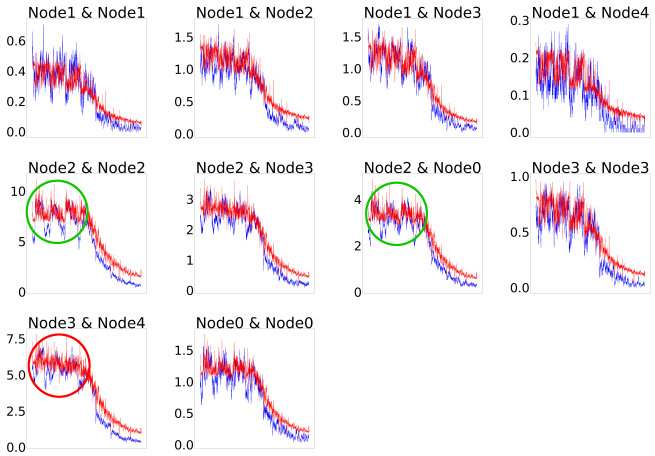

Fig. 7. Ground truth (blue) and predictions on test dataset (red) of the number of errors in the next time step for each node category pairs (1 second resampling). Y axis (please note, it has different scales): moving average of the number of errors in the next time step (window size: 100). X axis: time, duration of test data (from 16:48 to 23:59).

TABLE II
MEAN SQUARED ERROR OF THE NEURAL NETWORK'S PREDICTIONS ON TEST DATA FOR THE NODE CATEGORY PAIRS.

	Node1&Node1	Node1&Node2	Node1&Node3	Node1&Node4	Node2&Node2	Node2&Node3	Node2&Node0	Node3&Node3	Node3&Node4	Node0&Node0	Overall MSE
1 sec	0.2	0.7	0.7	0.1	6.4	1.8	2.3	0.4	4.3	0.8	1.8
10 sec	2.8	10.4	10.9	1.3	162.6	25.7	38.2	5.7	80.6	10.0	34.8

V. CONCLUSION

Applying DL techniques at various areas of telecommunications network and service management is promising. There are lots of space for improvement in terms of covering FCAPS management areas, which this paper provided suggestions for. As a specific example, we trained a deep convolutional neural architecture for predicting the number of errors for every node category pairs based on temporal features of the signaling in different timescales. In case of some nodes category pairs, the features were correlated to the output, however, the prediction was acceptable in case of other node category pairs. Based on these preliminary results by collecting more data and increasing the receptive field the prediction range is likely to be extensible and thus, errors on different parts of the network will be able to be predicted. Future work for this specific example will include training in greater data sets, and predicting for longer time periods (several minutes). Furthermore, other data sources of the VoLTE monitoring architecture is going to be introduced in the described DL infrastructure. This should allow the operator to act on the predictions in a timely manner.

VI. ACKNOWLEDGEMENTS

Parts of this research presented has been founded by the National Research, Development and Innovation Office, Hungary (KFI_16-1-2017-0024).

Parts of the research presented in this paper has been supported by the BME-Artificial Intelligence FIKP grant of Ministry of Human Resources (BME FIKP-MI/SC), by Doctoral Research Scholarship of Ministry of Human Resources (UNKP-18-4-BME-394) in the scope of New National Excellence Program, and by János Bolyai Research Scholarship of the Hungarian Academy of Sciences. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

We would like to thank Gábor Sey for his valuable insights.

REFERENCES

[1] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 16.

[2] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *arXiv preprint arXiv:1803.04311*, 2018. [Online]. Available: https://arxiv.org/pdf/1803.04311.pdf

[3] ITU-T, "Principles for a telecommunications management network," Rec. M3010, Oct. 1992.

[4] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, Jan 2018.

[5] O. Acker, A. Blockus, and F. Potscher, "Benefiting from big data: A new approach for the telecom industry," Strategy and PWC, April 2013.

[6] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self-organizing cellular networks," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2392–2431, Fourthquarter 2017.

[7] T. Shon and J. Moon, "A hybrid machine learning approach to network anomaly detection," *Information Sciences*, vol. 177, no. 18, pp. 3799 – 3821, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025507001648

[8] G. Kathareios, A. Anghel, A. Mate, R. Clauberg, and M. Gusat, "Catch it if you can: Real-time network anomaly detection with low false alarm rates," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2017, pp. 924–929.

[9] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 305–316.

[10] P. Varga, T. Tothfalusi, Z. Balogh, and G. Sey, "Complex solution for volte monitoring and cross-protocol data analysis," in *30th IEEE/IFIP Network Operations and Management Symposium (NOMS), AnNet*, Apr. 2018.

[11] T. Tothfalusi and P. Varga, "Assembling sip-based volte call data records based on network monitoring," *Telecommunications Systems*, vol. 68, no. 3, pp. 393–407, 2018.

[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: Session initiation protocol," RFC 3261, 2002.

[13] F. M. E. David C. Hoaglin and J. W. Tukey, *Understanding Robust and Exploratory Data Analysis*, 2000.

[14] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.