Anticipating Mobile Radio Networks Key Performance Indicators with Transfer Learning

Claudia Parera*[†][‡], Alessandro E. C. Redondi*, Matteo Cesana*, Qi Liao[†] and Ilaria Malanchini[†]
*DEIB, Politecnico di Milano, Milan, Italy
E-Mail: {claudia.parera, alessandroenrico.redondi, matteo.cesana}@polimi.it
[†]Nokia Bell Labs, Stuttgart, Germany
E-Mail: {qi.liao, ilaria.malanchini}@nokia-bell-labs.com
[‡]SnT, University of Luxembourg, Luxembourg
E-Mail: {claudia.parera}@uni.lu

***Abstract*—The use of artificial intelligence is foreseen to be pervasive in future mobile radio networks, enabling dynamic and proactive radio resource provisioning and allocation as well as end-to-end optimization of the network architecture. Current approaches in mobile radio networks commonly assume having a complete batch of data on the specific network element when optimizing and adapting the network working configuration. Such a pipeline is at odds with the increasing complexity and extreme flexibility of 5G and next generation systems where reconfiguration decisions might be taken rather frequently, and with only few data available. In this paper, we focus on the problem of predicting channel quality and average number of active user equipment when a limited amount of data is available from the cell to predict and a high number of predictions need to be carried out simultaneously. We propose a transfer learning framework based on one dimensional convolutional neural networks and explore several models with different complexity overhead for the prediction task across 100 cells. The performance of the proposed framework is validated against classical machine learning approaches in terms of accuracy and computation time when varying the amount of data available for training. Achieved results indicate that transfer learning outperforms the "non-transfer" approaches, specifically when the amount of data available from the cell to predict is scarce.**

***Index Terms*—Channel quality prediction, active users prediction, deep transfer learning, time series forecasting**

I. Introduction

Anticipating the network performance with high accuracy and low overhead can boost the proactive optimization of mobile radio networks. For instance, channel quality indicator (CQI) has been proposed for efficient resource allocation of video streaming traffic [1]. Similarly, cell load prediction, has been proposed to minimize network element sleeping time [2].

Traditional approaches to network performance prediction, leverage past information from the network node to predict. This is not always feasible, since the network layout is constantly evolving with the frequency roll out of new cells, therefore previous performance history is not available. Moreover, in current networks the number of cells to predict simultaneously is in the order of hundreds and is envisaged to grow in the future due to the current requirements of 5G networks and beyond.

This calls for new and efficient ways of predicting the network performance when limited information is available and the performance of multiple cells needs to be predicted simultaneously. To this end, we study network performance on 4G Long Term Evolution (LTE) cells operating at different frequencies (i.e. 1.8 GHz and 2.1 GHz) and located in several urban areas. In particular, we focus on CQI and cell load active number of user equipment (UE) Key Performance Indicators (KPIs), since they are among the most important KPIs metrics the operators could use to monitor the network status and optimizing accordingly. The CQI reflects the channel status experienced by the UEs to their respective cells, ranging from 0 to 15 in LTE. Throughout this paper, UE refers to the average number of active user equipment per cell during measurement period.

In this paper, we introduce a transfer learning framework to predict the CQI and UE for the different cells across the city. We design and test several strategies for picking candidate cells across the city for the transfer learning task.

The proposed framework can be conveniently used by a network operator to make educated decisions in a number of relevant situations:

- Network optimization/management: Only a sub-GHz frequency carrier is active at one cell, and the network operator needs to decide whether to activate higher frequency carriers by anticipating their expected quality.
- Radio resource/energy management: The higher frequencies carriers of cells perform duty cycling for energy management purposes, and the network operator needs to decide when to switch them on/off.

In a final step, we evaluate the performance of the proposed transfer learning approach against classical machine learning and statistical methods such as Autoarima. We use a dataset containing CQI and UE values from 100 cells collected over a time period of one month from a commercial mobile radio network deployed in a medium-sized city in Northern Italy.

In the following, we summarise the main contributions of this paper:

*At the time of writing this paper Claudia Parera was with Politecnico di Milano and Nokia Bell Labs.

- We provide deep learning architectures that significantly outperform statistical methods, capturing the high non-linearity of the CQI and UE.
- We introduce transfer learning to carry out predictions across different cells in the city. Our transfer learning method outperforms traditional machine learning methods when the amount of samples available from the target cells is limited.
- We significantly reduce the initial amount of training time with regard to approaches that do not use transfer learning, specifically when the number of cells to predict increases.
- Finally, we provide several strategies to select a set of candidate cells across the city to build a general model that can be transferred to the rest of the cells.

The rest of this paper is organized as follows: Section II reviews previous work in the area of CQI and UE predictions. Section III describes the reference dataset as well as the preprocessing steps followed. Section IV focuses on the proposed deep and transfer learning approaches. Numerical results and complexity analysis are reported in Section V. Finally, Section VI summarizes the main contributions of this paper and describes future research directions.

II. Related Work

Techniques to anticipate the network performance have been widely investigated in multi-fold network environments; either to take advantage of future link improvements or to counter bad conditions before they impact the system [3]. Techniques can be divided in two categories: (i) the ones that use traditional machine learning or statistical approaches and (ii) the ones that use deep learning.

In the former category, Wiener filters, cubic spline extrapolation and short-term average are used in [4] for CQI prediction. Other studies exploit the nonlinear characteristics of the channel. For instance, in [5] the spectrum sensing process is modeled as a non-stationary Hidden Markov Model. In [6], spatial and temporal correlation are taken into account to model the CQI prediction problem as a multivariate Gaussian Process. As for the cell load, similar kinds of studies have been carried out. In [7] the authors use k-Means and SVM classifiers on call detail record data to predict the cell load in order to improve network planning. Similarly, in [8], the authors use regression including spatio temporal features for cell load prediction.

In the second category, the major trend is modeling the prediction problem as a sequence problem using recurrent neural network (RNN) architecture variations that have been shown to be successful for this problem setting. As an example, in [9] Taguchi optimization, and long short-term memory networks (LSTMs) are used for spectrum prediction, specifically for channel quality as well as channel occupancy. In [10] KPI prediction is performed by stacking multiple LSTM building blocks together. Similar to our case, cell load and channel quality are used for results validation. Another approach has been proposed in [11] for 5G, where convolutional neural networks (CNNs) and LSTMs are proposed. For a comprehensive overview on channel quality prediction the interested reader may refer to the survey in [3]. The same kind of algorithms have also been used for cell load prediction. For instance, in [12] CNNs, LSTMs and the combination of both are used for traffic prediction evolution. As in our case, the authors comment on the complexity overhead of their approach.

In general, most of the aforementioned works leverage past information to anticipate future performance. Conversely, we target the case where limited data is available from the cell to predict and a high number of predictions needs to be carried out at the same time. Moreover, we show that selecting data from a “good candidate” set of cells can be used to carry out predictions on the rest of cells across the city with improved accuracy and decreased training time.

III. Problem Statement and Background

In this paper, we propose the application of machine learning algorithms to predict the channel quality and number of active users in 4G LTE wireless networks. The problem at hand can be defined as predicting future CQI or UE values when having limited data available from the cell to predict.

Let $(x_i)_{i=0}^{T-1} = \{x_0, x_1, \ldots, x_{T-1}\}$ be the sequence of CQI or UE values obtained for a target cell during T hours. The CQI and UE prediction problems can be formalized as follows: Given no or a limited amount (i.e. $t - l$) of CQI or UE observations from a target cell, $(x_i)_{i=l}^{t-1}$, we aim to forecast future CQI or UE values for the next hour $(\hat{y}_i)_{i=t}^{t+1}$. For this purpose, we leverage CQI or UE observations from different cells/frequency carriers chosen according to different selection criteria.

For the task at hand, we use a dataset containing past CQI and UE observations from a live 4G LTE network deployed in a medium-size city in Northern Italy. The reference dataset contains data from 100 cells working at 0.8, 1.8 and 2.1 GHz of frequency, respectively. Each time series element reports the hourly average of the CQI and UE. The total amount of data is equivalent to 583 CQI or UE measurements per time series. The data was recorded between January 8, 2017 and February 1, 2017, for a total of 24 days and 7 hours.

The following pre-processing steps were carried out:

- Missing value and outlier detection: No missing values and outliers were found.
- Stationary assessment: Most of the methods for time series forecasting work under the assumption that the time series is stationary. By using Dickey-Fuller [13] and KPSS [14] statistics tests, we found that in the majority of cases the data was already stationary. However, to avoid non-stationary cases, a first order difference transformation is carried out to the whole dataset.

IV. Prediction Approaches

In this section, we describe the proposed deep transfer learning method as well as the baselines used for benchmarking.

A. Deep Learning

Deep Neural Networks have significantly improved the state of the art in a variety of fields such as Computer Vision or Natural Language Processing. For instance, CNNs are powerful feature extractors for hierarchical data since the lower layers of the network capture more general patterns, whereas the deeper layers extract the more specific ones. Inspired by this fact, we use CNNs as the building blocks of our transfer learning pipeline.

The deep learning pipeline consists of the following steps:

1) Preprocessing: First, the general time series forecasting problem is re-framed as a supervised machine learning problem. For this purpose, we use sliding windows of size $w = 24$ by shifting the original time series one step to the right T times. Figure 1 shows the process in detail. The resulting supervised machine learning problem is defined as finding the function $g(\mathbf{x}, \boldsymbol{\theta}) = y$, $\mathbf{x} \in \mathbb{R}^w$ and $y \in \mathbb{R}$ that maps w hours of CQI or UE observations to the CQI or UE value on the next hour. Each input vector $\mathbf{x}$ is given by the sub-sequence $(x_i)_{i=t-1}^{t-w}$, y is given by y_t and $\boldsymbol{\theta}$ represent the neural network weights. Then the data is divided into training, cross validation and test sets. Before training, the data is scaled between -1 and 1 by using a min-max scaler, which is fitted to the training set and applied to the cross validation and test sets.
2) Training: During training, a model $g(\mathbf{x}, \boldsymbol{\theta})$ is created by fitting the selected architecture on the training set.
3) Cross Validation: The cross validation set is used at a later stage by the network to select a good combination of hyperparameters.
4) Testing: At testing time, $g(\mathbf{x}, \boldsymbol{\theta})$ is applied to data coming from the same or a different frequency carrier. The data is projected onto the original space by reversing the scaling and first order difference transformations. It is worth noting that the first order difference and scaling transformations should be reversed before evaluating the performance of our algorithms.
5) Performance Evaluation: The prediction accuracy is measured by the root mean squared error (RMSE) between real and predicted values in the test set (with size $T_{\text{test}} = N$) defined by Eq. 1:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=t}^{t+N}(\hat{y}_i - y_i)^2}. \quad (1)$$

Figure 2 shows the one dimensional (1D) CNN architecture proposed in this work. It is comprised of 7 layers; the first 5 layers of the network are a combination of 1D convolutional layers followed by a 1D max pooling layer. Finally, a flatten layer as well as a dense layer are stacked for producing the final output. We apply rectified linear unit (ReLU) nonlinear transformations as the activation function. We use 256 filters and a kernel size of 3. For the max pooling layers we use stride $s = 2$ and pool size $\delta = 2$. The dimension of the first convolutional layer corresponds to the dimension of the feature space, which is the window size $w = 24$, in our case.

B. Transfer Learning

Given a source domain D_S with enough data and a target domain D_T with limited data, for the transfer learning task on D_T, we first train a model M_S on D_S. Then, we create a new model M_T by taking the previous model M_S, freezing its first layers and adding new layers. The weights on the new layers of M_T can be either randomly initialized or initialized with M_S weights. Finally, we retrain M_T on the available data from D_T. As an example, we show in Figure 2 a CNN architecture, where we freeze the first 2 convolutional layers and randomly initialize and retrain the last convolutional layer. The idea behind this is transferring the more general features learned by the first layers of the network in a richer D_S (i.e. longer time series) to a limited D_T (i.e. shorter time series). For comparison fairness we use the same architecture for M_S and M_T.

We developed and tested several transfer learning methods. They differ in the way D_S cells are chosen to train M_S.

1) Transfer Frequency Model (CNN TFM): D_S contains one cell at a lower frequency in the same geographical location.
2) Transfer Random Model (CNN TRM): D_S contains 5 cells chosen randomly across the city.
3) Transfer City Model (CNN TCM): First, we cluster all the cells across the city to identify similar groups of cells. Then, we choose D_S cells as the closest cell to the centroid of each cluster in order to have the model as "representative" of the whole city as possible. In Section IV-C, we give more details on the clustering approach and distance metric to select the closest cell.
4) Transfer Cluster Model (CNN TCLM): We also cluster all the cells across the city. Then, we train a model per cluster by choosing D_S cells as the 5 closest cells to the centroid of each cluster. For instance, if the number of clusters is 2, we will have 2 source models, M_{S_1} and M_{S_2}, trained on cells from clusters 1 and 2, respectively. For the prediction task on D_T, if a cell is in cluster C, with $C = 1$ or $C = 2$, then M_{S_C} is chosen as source model.
5) Transfer Hybrid Model (CNN THM): Similar to CNN TCM. In addition, we embed the cluster information as an input feature to the model in a different CNN channel. The cluster label is a categorical variable added as one-hot encoding [15].
6) Transfer Hybrid Model with Correlated Counters (CNN THM CC): Similar to the CNN TCM but instead, using domain knowledge, we add the correlated counters as input features to the model. Each correlated counter represents a new feature in a different CNN channel.

Below we use the term city models to denote CNN TRM, CNN TCM, CNN TCLM, CNN THM and CNN THM CC, since these models use information from other cells across the city for transfer learning.

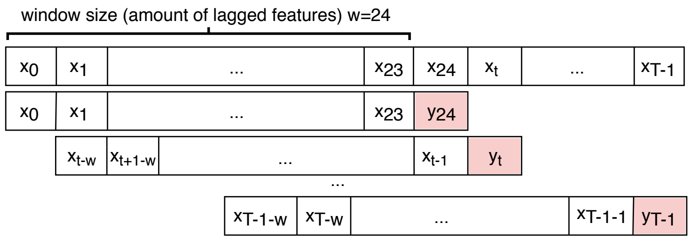


Fig. 1: Sliding windows for multi-step time series forecasting

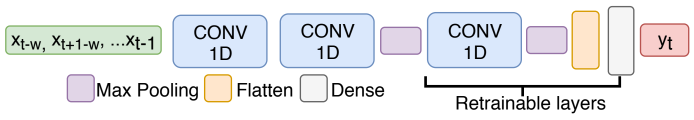


Fig. 2: CNN architecture

C. Clustering and Distance Metric

CNN TCM, CNN TCLM and CNN THM use cell cluster information as shown before. We identify the different cell clusters across the city by grouping the raw time series that represent each cell. This is done for each KPI independently. We use k-Means [16] with the Euclidean Distance, since we are interested in clustering based on the similarity of time series data without taking time delay into account. After trying different values of k (number of clusters), we chose 2 and 5 as the number of clusters. CNN TCLM 2 and CNN TCLM 5 use 2 and 5 clusters, respectively.

D. Baselines

As baselines, we use traditional time series forecasting methods such as Autoarima and two CNNs methods, without using transfer learning. By selecting Autoarima, as baseline, we assess whether a deep neural network should be trained or linear methods such as Autoarima are sufficient for prediction task at hand. If deep learning methods achieve better prediction error, we compare the performance of transfer learning models (see Section IV-B) against the performance of similar models trained on data from D_T (i.e., CNN S) or trained on data from D_S and applied to D_T without retraining (i.e. CNN BS). The goal is assessing whether *negative transfer* will happen and the retraining step in transfer learning is really needed. The baseline methods are described as it follows:

1) Autoarima: Auto-Regressive Integrated Moving Average (ARIMA) introduced by Jenkins in [17]. Seasonal ARIMA models are usually denoted by ARIMA$(p, d, q)(P, D, Q)_m$, where coefficients p, d, q are the order of the autoregressive model, the degree of differencing, and the order of the moving-average model, respectively. m refers to the number of periods in each season and P, D, Q refer to the autoregressive, differencing, and moving average terms of the seasonal part of the model, respectively. By using grid search we optimize the ARIMA coefficients (p, d, q, m, P, D, Q) that fit our data.
2) CNN S: We train a model with limited data from the same cell in D_T.
3) CNN BS: We train a model with enough data from a different cell in D_S and use it to carry out predictions on D_T without the retraining step on D_T.

V. Numerical Results

In this section, we first describe the experimental setup and parameters optimization for the source and target models. Then, we show the performance achieved by the transfer learning method in terms of accuracy, complexity and training times.

A. Experimental Setup

The reference dataset contains 100 time series per KPI. Each time series contains 583 samples with 1 h of time granularity (see Section III). Every time series is divided into training, cross validation and test sets containing 535, 24 and 24 samples, respectively. For each of the algorithms tested, we show the average RMSE across cells when changing the amount of days taken from D_T for training a model without transfer (i.e., CNN S), or for retraining a model as part of the transfer learning pipeline (i.e., CNN TFM, CNN TRM, CNN TCM, CNN THM, CNN TCLM 2, CNN TCLM 5, CNN THM CC).

Experiments were performed on a PC with three Intel Xeon E5 v3/v4 CPUs, one GeForce RTX 2080Ti GPU card and 16 GB of RAM.

B. Parameter Optimization

The different prediction approaches encompass different parameters, which require fine tuning for further optimization. Autoarima requires $m = 24$ to be set *a priori* (Section V-C). For the CNNs, we use $w = 24$ h to predict the next hour. We fix the batch size to 128, manual cross validation is carried out in order to choose a good architecture for this problem. The sections below show the hyperparameters selection process for M_S and M_T, respectively.

Hyperparameter	Default Values
Lag Number Input	24
Number of Epochs	300
Learning Rate	0.001
Number of Layers	3
Number of Filters	256
Dropout	0

TABLE I. Default combination of hyperparameters

1) M_S Parameter Optimization: In order to find the best architecture for the CQI and UE prediction problems we begin with the default configuration shown in Table I. We try different combinations of hyperparameters using all available data (i.e. 22 days). We choose the setting that performs the best on average for the 100 available cells. Below we show the different values tried and highlight the best configuration for each step (see Tables II, III IV, V, VI and VII below for more details). We can conclude that less than 300 epochs and a learning rate smaller than 0.001 improves performance for both KPIs. For the CQI a smaller architecture leads to better result, whereas for UE a more complex architecture helps reducing the error as well as adding 0.2% of dropout.

	Lag input number		
KPI	24	48	96
CQI	**1.70**	1.74	1.74
UE	**1.80**	1.83	1.80

TABLE II. RMSE per lag input number

	Number of epochs		
KPI	100	300	500
CQI	**1.64**	1.70	1.69
UE	1.81	**1.80**	1.83

TABLE III. RMSE per number of epochs

	Learning rate			
KPI	0.0001	0.001	0.01	0.1
CQI	**1.40**	1.64	1.57	1.75
UE	1.84	**1.80**	2.08	3.31

TABLE IV. RMSE per learning rate

	Number of layers		
KPI	2	3	4
CQI	**1.30**	1.40	1.44
UE	1.94	1.80	**1.71**

TABLE V. RMSE per number of layers

	Number of filters		
KPI	128	256	512
CQI	1.32	**1.30**	1.32
UE	1.71	**1.71**	1.73

TABLE VI. RMSE per number of filters

	Dropout percentage		
KPI	0	0.2	0.5
CQI	**1.30**	1.30	1.31
UE	1.71	**1.68**	1.89

TABLE VII. RMSE per dropout percentages

2) M_T Parameter Optimization: To find the best architecture for M_T, we use the same architecture as M_S to ensure comparison fairness. We optimize the following parameters in M_T:

- Number of frozen and retrainable layers: The amount of layers to freeze or retrain on M_T largely dictates the amount of knowledge to transfer from D_S to D_T. Depending on the data, if the number of layers to retrain is not the "optimum", *negative transfer* can happen.
- Weight initialization: The weights in the retrainable layers of M_T, can be either randomly initialized or taken from M_S. This also affects the amount of knowledge transferred from D_S to D_T.

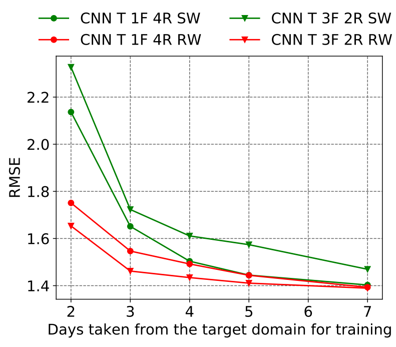
We tried all possible combinations of layers to freeze and retrain, as well as random and non random initialization. We carried out this process on CNN TCM, where just one model is trained for D_S, since doing this for every transfer learning model in D_S and D_T would be extremely time consuming (more details about training times can be found in Section V-E). Figure 3 shows the results for the different days taken from D_T. Results indicate that for CQI the best transfer learning model is that which has the first 3 layers frozen, and the last 2 are retrained. Overall, using random initialization leads to improved performance. In contrast, we can observe that for the UE KPI, freezing a high number of layers, which means transferring more knowledge from D_S, worsens the performance. There is no significant difference between using random initialization, or taking the weights from D_S. This is expected since a higher number of layers need to be retrained on M_T.

C. City Models Transfer

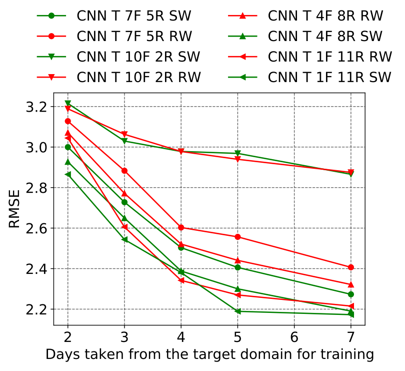
In this section we show the performance of the transfer learning approach for the available 100 cells. Figure 4 shows the average RMSE when applying the different models (see Section IV-B for more details about the different models). The x-axis shows the amount of samples (measured in days) available from the target domain that was used for training or fine tuning the models accordingly. The S curves refer to cases where the proposed methods are applied by leveraging only data available from the same cell. The TL curves are related to the transfer learning scenarios, where the model is pretrained on D_S, then retrained on D_T.

We can draw the following conclusions:

- All the deep learning methods, whether they use transfer learning or not, outperform traditional forecasting methods such as Autoarima. This different is slightly more significant when a limited amount of data is taken from D_T.
- For the CQI, the transfer learning methods outperform CNN S, which does not use transfer learning. For UE CNN TRM, CNN TCM and CNN THM outperform CNN S.
- For CQI, CNN TCLM 2, CNN TCLM 5 and CNN THM are the models that perform the best according to RMSE values. In contrast, for UE, CNN THM CC is the model with the best performance.

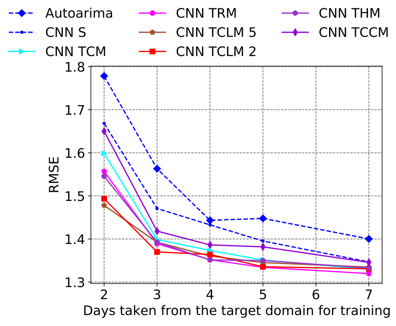


(a) RSME CQI

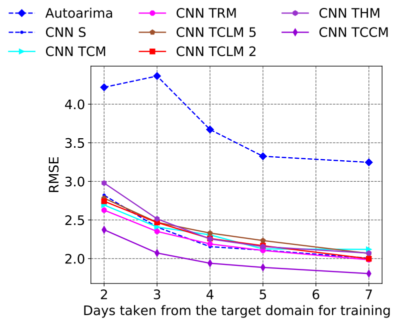


(b) RSME UE

Fig. 3: M_T Parameter Optimization



(a) RSME CQI



(b) RSME UE

Fig. 4: City Models

D. *Frequency vs. City Models Transfer*

In this section, we compare the city models performance with the CNN TFM performance. The CNN TFM requires that data from the same location, at a higher frequency, as such, it cannot be used when the higher frequency is turned off. More information is available in Section IV-B. Figure 5 shows the average RMSE over 100 cells when the higher frequency is active. We can observe the following:

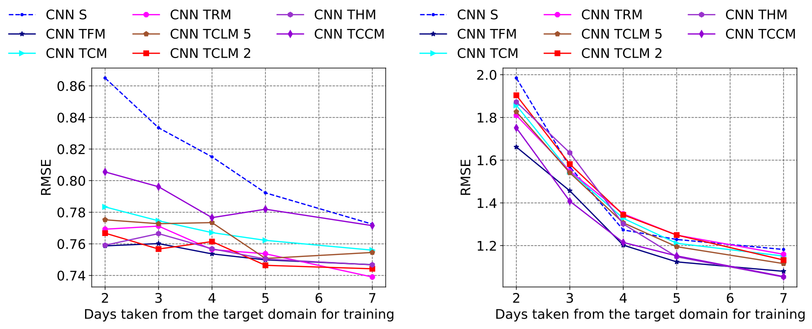
- In cells where both layers are active CNN TFM achieves comparable performance to CNN TCLM 2 for CQI and CNN THM CC for UE.
- In addition, the amount of models to train when using CNN TFM to carry out predictions on n cells from D_T is $2 * n$, whereas with the city model approaches this number decreased to $5+n$ in the worse case (see Section V-E for more details on model complexity).

E. *Complexity Analysis*

We compare all the approaches in terms of amount of parameters to find and training times.

1) Trainable Parameters: Table VIII shows the total amount of parameters found during training by each of the deep learning approaches with and without transfer learning. The lower the amount of parameters to find, the lower the training time. We use $n = 100$ to denote the number of cells, and c to denote the number of clusters when using CNN TCLM. In Table VIII, it can be observed that CNN TRM, CNN TCM, CNN THM and CNN THM CC are the methods with the smallest number of trainable parameters to be found since, for each of them, just one model is trained on D_S.

2) Cell Training Time: Given n cells to predict, training, retraining and inference can be carried out either in parallel or sequentially. In this section, we show the training and retraining times per cell, operations are carried out in parallel. For transfer learning, we first train M_S on D_S and after that, for each of the n cells on D_T, we create n M_T models that are retrained at the same time. Once the retraining step is completed, we use the n M_T models to carry out predictions simultaneously. If we do not use transfer learning (i.e., CNN S) we only train n models on D_T at the same time. We believe that the possibility of training and retraining different models in parallel is one of the strengths of our transfer learning approach, specifically for use cases where there is a central entity and retraining can be performed per network node at the same time. Figures 6a and 6b show the average training time without transfer learning (i.e., CNN S), the initialization step of transfer learning by training a model on D_S (i.e., CNN SCM, CNN SRM, CNN SCLM, CNN SHM, CNN SRM) and during the retraining step of transfer learning (i.e., CNN TCM, CNN TRM, CNN TCLM, CNN THM, CNN TRM). We can



(a) RSME CQI (b) RSME UE

Fig. 5: Frequency vs. City models

Model	Total Parameters (CQI)	Total Parameters (UE)
CNN S	$n * 20,049 = 20,044,900$	$n * 591,873 = 59,187,300$
CNN TFM	$n * 200,449 + n * 2,561 = 20,301,000$	$n * 591,873 + n * 590,849 = 118,272,200$
CNN TRM	$1 * 200,449 + n * 2,561 = 456,549$	$1 * 591,873 + n * 590,849 = 59,676,773$
CNN TCM	$1 * 200,449 + n * 2,561 = 456,549$	$1 * 591,873 + n * 590,849 = 59,676,773$
CNN TCLM 2	$2 * 200,449 + n * 2,561 =$ 656,998	$2 * 591,873 + n * 590,849 = 60,268,646$
CNN TCLM 5	$5 * 200,449 + n * 2,561 =$ 1,258,345	$5 * 591,873 + n * 590,849 = 62,044,265$
CNN THM	$1 * 204,289 + n * 2,561 = 460,389$	$1 * 595,713 + n * 590,849 = 59,680,613$
CNN THM CC	$1 * 205,825 + n * 2,561 = 461,925$	$1 * 598,785 + n * 590,849 = 59,683,685$

TABLE VIII. Trainable parameters

make the following observations:

- Training times for CQI are lower than for UE as the network architecture is smaller.
- In Figure 6a the transfer learning retraining step (for CNN TCM, CNN TRM, CNN TCLM, CNN THM, CNN TRM) is 1 second faster than training a model without transfer learning (i.e., CNN S). This is reasonable since during the retraining step we freeze the first layers of the model and we just need to find parameters for the retrainable layers, which are less. The retraining step for the UE (see Figure 6b) is less than a second faster as we freeze less layers when retraining on D_T.
- There are no noticeable differences between the transfer learning models in terms of retraining time per cell.
- The total transfer learning training time, which is comprised of training time on D_S plus the retraining time on D_T with limited data, is going to be considerably higher than not using transfer learning. However, the training step on D_S can be performed in the best case only once, or in the worst case significantly less often than retraining, depending on the use case requirements.

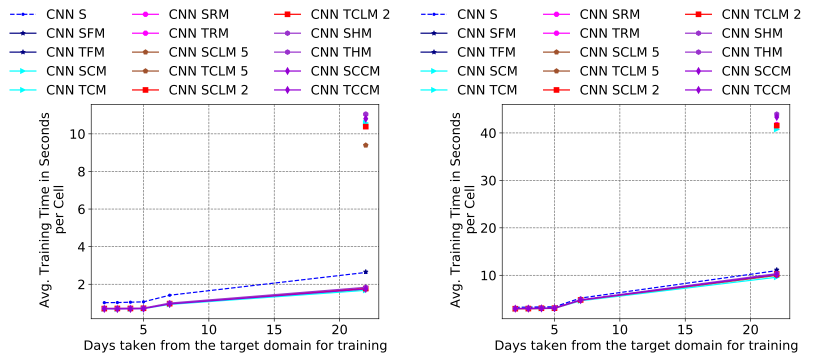
3) Total Training Time: If predictions are carried out sequentially, training, retraining and inference per cell are performed one after each other. In Table IX we show the total training time if predictions are performed for a batch of 100 cells taking 7 days data from D_T. We can conclude:

- The transfer learning approaches, specifically the city models (i.e., CNN TCM, CNN TRM, CNN TCLM, CNN THM, CNN THM CC) are faster than the CNN S, which does not use transfer. However, this is not the case for CNN TFM, since for each cell on D_T a model has to be trained on D_S.
- Among the transfer learning approaches, CNN TCM, CNN TRM and CNN THM CC are the fastest models for both KPIs, with training times around 3 and 16 minutes, respectively.
- We are able to make predictions for 100 cells with averages of 2.95 minutes for CQI, and 16.66 minutes for UE, when training on D_S every time a prediction is carried out on D_T. If we just consider the retraining time we can make predictions for 100 cells with averages of 2.78 minutes and 15.98 minutes.
- In terms of scalability, higher gains are expected when the number of cells to predict is considerably higher than 100. For instance, considering $n = 30000$ for CQI, using CNN S the total training time would be 21.83 hours, whereas using CNN TCM for the same amount of cells the training time would be 13.91 saving 8 hours of training.

VI. Conclusions

We proposed a transfer learning framework designed to predict CQI and UE in the challenging case where the amount of data available from a given cell is limited, and a high number of predictions need to be carried out in a short period of time. The proposed framework was tested on a dataset from a commercial 4G LTE network, showing how transfer learning can be carried out across cells working at different frequencies, or situated in different locations around a city.

The obtained results show that the proposed deep transfer learning methods are particularly effective in terms of training time when the amount of cells to predict is high. Therefore, we can conclude there are several advantages in terms of



(a) Duration CQI, D_T (b) Duration UE, D_T

Fig. 6: Training Times per Cell in Seconds

Model	Total Training Time in Minutes (CQI)	Total Training Time in Minutes (UE)
CNN S	$n * 2.62s = 4.36m$	$n * 10.98s = 18.3m$
CNN TFM	$n * 2.66s + n * 1.76s = 7.36m$	$n * 11.14s + n * 10.24s = 35.63m$
CNN TRM	$1 * 11.04s + n * 1.77s = 3.13m$	$1 * 41.77s + n * 9.99s = 16.67m$
CNN TCM	$1 * 10.57s + n * 1.67s = 2.95m$	$1 * 40.74s + n * 9.59s = 16.66m$
CNN TCLM 2	$2 * 10.38s + n * 1.77s = 3.29m$	$2 * 41.55s + n * 10.27s = 18.50m$
CNN TCLM 5	$5 * 9.39s + n * 1.74s = 3.18m$	$5 * 41.49s + n * 9.97s = 20.06m$
CNN THM	$1 * 11.04s + n * 1.83s = 3.23m$	$1 * 43.95s + n * 10.35s = 17.96m$
CNN THM CC	$1 * 10.80s + n * 1.75s = 3.08m$	$1 * 43.35s + n * 10.14s = 17.62m$

TABLE IX. Sequential training time

performance and scalability on using transfer learning for network performance prediction. Future work will include the application of the proposed algorithms to predict a longer lookahead in the future and with a higher number of cells.

Acknowledgement

This work has been partially funded by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 643002. The authors would also like to thank the 360Lab, University of Luxembourg for its financial contribution.

References

[1] D. Tsilimantos, A. Nogales-Gómez, and S. Valentin, "Anticipatory radio resource management for mobile video streaming with linear programming," in *IEEE ICC*, 2016, pp. 1–6.

[2] R. Li, Z. Zhao, X. Zhou, and H. Zhang, "Energy savings scheme in radio access networks via compressive sensing-based traffic load prediction," *Transactions on emerging telecommunications technologies*, vol. 25, no. 4, pp. 468–478, 2014.

[3] N. Bui, M. Cesana, S. A. Hosseini, Q. Liao, I. Malanchini, and J. Widmer, "A survey of anticipatory mobile networking: Context-based classification, prediction methodologies, and optimization techniques," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1790–1821, 2017.

[4] X. Xu, M. Ni, and R. Mathar, "Improving QoS by predictive channel quality feedback for LTE," in *IEEE SoftCOM*, 2013, pp. 1–5.

[5] X. Xing, T. Jing, Y. Huo, H. Li, and X. Cheng, "Channel quality prediction based on bayesian inference in cognitive radio networks," in *IEEE INFOCOM*, 2013, pp. 1465–1473.

[6] Q. Liao, S. Valentin, and S. Stańczak, "Channel gain prediction in wireless networks based on spatial-temporal correlation," in *IEEE SPAWC*, 2015, pp. 400–404.

[7] S. E. Hammami, H. Afifi, M. Marot, and V. Gauthier, "Network planning tool based on network classification and load prediction," in *2016 IEEE Wireless Communications and Networking Conference*. IEEE, 2016, pp. 1–6.

[8] E. Lovisotto, E. Vianello, D. Cazzaro, M. Polese, F. Chiariotti, D. Zucchetto, A. Zanella, and M. Zorzi, "Cell traffic prediction using joint spatio-temporal information," in *2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2017, pp. 1–4.

[9] L. Yu, J. Chen, G. Ding, Y. Tu, J. Yang, and J. Sun, "Spectrum prediction based on Taguchi method in deep learning with long short-term memory," *IEEE Access*, vol. 6, pp. 45 923–45 933, 2018.

[10] A. Bhorkar, K. Zhang, and J. Wang, "DeepAuto: A hierarchical deep learning framework for real-time prediction in cellular networks," *arXiv preprint arXiv:2001.01553*, 2019.

[11] C. Luo, J. Ji, Q. Wang, X. Chen, and P. Li, "Channel state information prediction for 5G wireless communications: A deep learning approach," *IEEE Transactions on Network Science and Engineering*, 2018.

[12] I. Alawe, Y. Hadjadj-Aoul, A. Ksentinit, P. Bertin, C. Viho, and D. Darche, "An efficient and lightweight load forecasting for proactive scaling in 5G mobile networks," in *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2018, pp. 1–6.

[13] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American statistical association*, vol. 74, no. 366a, pp. 427–431, 1979.

[14] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?" *Journal of econometrics*, vol. 54, no. 1-3, pp. 159–178, 1992.

[15] M. A. Hardy, *Regression with dummy variables*. Sage, 1993, no. 93.

[16] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[17] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.