POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

TRANSFER LEARNING FOR ANTICIPATING MOBILE NETWORK PERFORMANCE

Doctoral Dissertation of:
Claudia Parera

Supervisor:
Prof. Matteo Cesana

Tutor:
Prof. Carlos Giussepe Riva

The Chair of the Doctoral Program:
Prof. Matteo Cesana

Year 2019 – Cycle 32

Acknowledgements

I would like to thank the following people for their continuous support throughout this project:

My main supervisor Prof. Matteo Cesana from Politecnico di Milano for his patience, motivation and the fruitful technical discussions.

My other supervisors Alessandro Redondi from Politecnico di Milano, Ilaria Malanchini and Qi Liao from Nokia Bell Labs, Stuttgart for their continuous guidance and feedback throughout this project.

Markus Gruber and the rest of the Radio Network Automation team at Nokia Bell Labs, Stuttgart who, in one way or another, supported me during my two years internship at Bell Labs.

Dan Wellington, Arto Sakko and Praveen Bezawada from Nokia Software for the insightful technical discussions and their willingness to collaborate.

Karin Wrobel who made me feel at home in Germany for the last two years.

My parents, especially my mum, for their patience and encouragement. My parents in laws for their support and care.

My husband Robert Norvill for his unwavering support and love.

Abstract

MACHINE learning will play a major role in handling the complexity of future mobile wireless networks by enhancing network management and orchestration capabilities. Due to the large number of parameters that can be configured in the network, collecting and processing high volumes of data is often unfeasible during network runtime. This calls for taking resource management and service orchestration decisions when only a partial view of the network is available, and multiple decisions need to be taken within a limited period of time.

Transfer learning is a machine learning paradigm that aims at improving the prediction performance of a learning task by applying knowledge previously gained in a related learning task or domain. Motivated by this fact, in this thesis we provide a transfer learning framework that can be applied to anticipate and further adapt network decisions when only a partial network view is available. Predictions can be carried out with improved performance when information in the target network domain is limited, and multiple decisions need to be made in a small time frame. To this end, we evaluate the proposed framework in three different industry provided, real world use cases, using data collected from commercial 4G networks. Namely, they are: (i) Tilt-Dependent Radio Map Prediction, (ii) Mobile Radio Networks Key Performance Indicator Anticipation and (iii) Multi-Step Resource Utilization Prediction.

The main contribution of this thesis is the introduction of transfer learning to anticipate network performance in mobile communication networks achieving improved accuracy and complexity time.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The fifth generation wireless networks (5G) are expected to improve the performance of cellular systems, achieving higher data rates, reduced latency, higher reliability and support for a greater numbers of users. To achieve this, 5G makes use of dense and heterogeneous deployments, coupled with higher flexibility in the network access and core domains, which can be dynamically managed in either a centralized or distributed manner.

To cope with such a challenging scenario, it is foreseen that machine learning tools will play a major role in enabling the transition from current mobile networks to 5G architectures [46]. Moreover, by exploiting the increased availability of data in 5G coming from network devices and user terminals, machine learning tools will be able to assist network operators in dealing with the increasing complexity of configuring parameters for network optimization. Thus, machine learning tools will form the basis for automated and smart network management techniques. Machines learning faces two major challenges in this regard: i) Data availability and ii) Time complexity.

- **Data availability**: Although the amount of data available is expected

to increase in 5G networks and beyond, collecting and storing high volumes of data with the required quality is often unfeasible for the operator.

For instance, conventional "drive tests" are carried out to collect data and improve network coverage and quality of service (QoS). However, this requires large Operational Expenditure (OPEX) and provides only limited snapshots of the entire network [40]. Other strategies such as crowdsourcing, can be effective, however they still raise questions in terms of security, privacy and trust [30].

In addition, mobile networks are constantly changing and evolving, which means the operator is constantly taking decisions to optimize network performance, making it impossible to collect and store data from all the possible network configurations at all time intervals. For instance, in [10], cells are turned off for energy saving purposes based on traffic demand, therefore no data can be collected when the cells are turned off.

Similarly, data quality problems, such as higher variance or sparsity can arise when data collection is carried out at different network layers, since propagation characteristics of components carriers working at different frequencies varies significantly. For instance, a component carrier working at 800 MHz has different propagation to a component carrier working at 2.5 GHz [59].

- **Time complexity**: To fulfill the requirements of 5G networks and beyond, fast and scalable anticipation and network optimization decisions are of vital importance. For instance, when carrying out Key Performance Indicator (KPI) anticipation, which is an essential task to enhance self organizing networks (SON) capabilities, thousands of predictions need to be carried out simultaneously for all the active cells in the system within a given timeframe. As a result, the main goal is not just achieving a "good" prediction error but also decreasing the prediction time, such that the system is capable of reacting on time (e.g., fault detection and mitigation as part of SON self-healing functionalities). Moreover, it is expected that in such a system, the number of cells will increase in the future, therefore scalability is another key requirement in order to anticipate the network performance and react accordingly.

  Similar requirements are encountered when carrying out network planning in order to handle network capacity, specifically for decentralized architectures, where a first processing step can be performed in

a central unit and minimal post processing can be carried out on a distributed manner on the network nodes.

The evolving 5G ecosystem calls for rethinking the way anticipation and optimization decisions are taken when only a partial view of the network is available and multiple decisions need to be taken simultaneously. To this end, in this thesis, we introduce and propose transfer learning, which is a machine learning paradigm that has received a lot of attention over the past few years, specifically in the industry [67]. Transfer learning seeks to improve the performance of a learning task by using knowledge gained in another related but different learning task or domain. Recently, transfer learning has been used with great success for use cases where a model is trained with sufficient data from a source domain and retrained and applied to another related task or domain with limited data, significantly improving the performance on the target task [50, 69].

1.2 Contributions

The main contributions of this thesis are summarized as it follows:

1. We propose the use of transfer learning for network performance anticipation by designing and developing a deep transfer learning framework that can be used when only a partial view of the network is available and multiple predictions need to be taken within a limited period of time.

2. In particular, the proposed transfer learning framework has been validated on real world data collected from Long Term Evolution (LTE) networks. The framework was built to solve three industry problem, namely they are: (i) Tilt-Dependent Radio Map Prediction, (ii) Mobile Radio Networks Key Performance Indicator Anticipation and (iii) Multi-Step Resource Utilization Prediction.

3. For all the tested scenarios, the proposed approach is shown to achieve state of the art results in terms of prediction error while significantly reducing the amount of labeled data required to carry out predictions.

4. For each use case, we answer the main transfer learning research questions such as: *What, When and How to transfer?* We accomplish this firstly by: i) Finding a neural network architecture and combinations of hyperparameters which improves the learning task on the target domain and ii) Selecting the source domain based on predefined similarity metrics tailored to the problem at hand.

5. Finally, by using transfer learning we significantly reduce the total amount of parameters that need to be found during training, thus decreasing the overall time complexity and increasing scalability.
6. The transfer learning solution has been adopted for integration on the Nokia portfolio on new use cases such as cell congestion and network capacity prediction. It will form part of Nokia's decision making process in the future.

1.3 Thesis Outline

The rest of this thesis is divided in chapters as it follows:

- Chapter 2, summarizes the state of the art on transfer learning as well as its applications in the area of mobile wireless networks.
- Chapter 3, formalizes transfer learning and describes the main components of the proposed transfer learning framework.
- Chapter 4, 5 and 6 describe different use cases in mobile network communications where transfer learning has been successfully applied.
- Chapter 7 concludes the work and summarizes further research directions.

1.4 Publications

The content in chapters 4, 5 and 6 has been submitted and accepted to international peer reviewed journals and conferences. Below the detailed list of authors, publications and venues:

- C. Parera, A. E. C. Redondi, M. Cesana, Q. Liao, L. Ewe, C. Tatino, "Transferring Knowledge for Tilt-Dependent Radio Map Prediction", IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, Spain, 2018
- Parera, C., Liao, Q., Malanchini, I., Tatino, C., Redondi, A.E. and Cesana, M., 2020. Transfer Learning for Tilt-Dependent Radio Map Prediction. IEEE Transactions on Cognitive Communications and Networking
- C. Parera, A. E. C. Redondi, M. Cesana, Q. Liao, I. Malanchini, "Transfer Learning for Channel Quality Prediction", IEEE International Symposium on Measurements and Networking, Catania, Italy, 2019

- C. Parera, Q. Liao, I. Malanchini, "Anticipating Mobile Radio Networks Key Performance Indicators with Transfer Learning" (Pending Submission)
- C. Parera, Q. Liao, I. Malanchini, D. Wellington, A. E. C. Redondi, M. Cesana, "Transfer Learning for Multi-Step Resource Utilization Prediction", IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, London, 2020 (Submitted)

CHAPTER 2

State of the Art

In this chapter we review the state of the art for transfer learning with particular focus on its application to mobile wireless networks.

2.1 Transfer Learning

Transfer learning has been applied with great success to supervised, unsupervised and reinforcement learning. Three seminal papers review the state of the art of transfer learning and its applications in those areas [74,93,102], respectively. More recently, deep transfer learning has gained a lot of attention [67], due to its success in Computer Vision (CV) and Natural language processing (NLP), among others. Main works in the area of deep transfer learning are reviewed in [91].

The literature related to transfer learning uses various terms interchangeably, such as: *covariance shift*, *bias* and *domain adaptation*. In this thesis, we follow the definition of transfer learning given in [74], where transfer learning aims at extracting knowledge from one or more source tasks and applies the extracted knowledge to a target task; allowing the domains, tasks and distributions of source and target domains being different (see Section 3.1 for a formal definition of transfer learning).

The main research questions when dealing with a transfer learning problem, are: *When to transfer?*, *What to transfer?* and *How to transfer?*. In the sections below we explain and show through examples how these questions have been addressed in the literature.

2.1.1 When to Transfer?

The question *When to transfer?* is mainly related to the issue of avoiding *negative transfer*, which happens when transfer learning has a negative impact on the performance of the target learning task.

The literature is primarily focused on the first two questions. In contrast, in this thesis, we show through the examples in Chapters 4, 5 and 6, the scenarios where transfer learning is carried out successfully and *negative transfer* is avoided.

2.1.2 What to Transfer?

For the purpose of studying *What to transfer?*, we will use the categorization found in [74], where transfer learning can be divided into:

- **Inductive transfer:** Different source and target tasks and same or different source and target domains.
- **Transductive transfer:** Same source and target tasks but different source and target domains.
- **Unsupervised transfer:** Similar to inductive transfer, with different but related source and target tasks. The focus is solving an unsupervised learning task on the target domain. There is no labeled data available from source and target domains during training.

Following this categorization, our work is an example of transductive transfer learning. We always have the same learning problem for the source and target tasks. Moreover, we assume that the distributions of the data in source and target domains are "related" but not exactly the same.

2.1.3 How to Transfer?

To answer the question *How to transfer?*, the most common transfer learning approaches are:

- **Instance transfer:** Labeled samples in the source domain are reweighted and used in the target domain. It can be applied to inductive and transductive learning.

In [21] the authors propose TrAdaBoost, which is the transfer learning version of AdaBoost [32]. Assuming the data distribution in training and testing domains are different, TrAdaBoost re-weights the samples in the source domain in an iterative manner. This reduces the effect of the "bad" instances and empowers the effect of the "good" ones in the final classification.

A similar approach is used in [47], where the authors use heuristics to remove instances from the source domain in order to minimize the difference between the conditional probabilities in source and target domains.

In contrast, in [14] the authors correct the differences between the conditional probability distribution in source and target domains without removing instances. They develop CP-MDA and 2SW-MDA algorithms assuming a limited amount of labeled data in the target domain. This extends [27], where source instance weights are defined as a function of the conditional probability distribution.

- **Feature transfer:** Aims at finding a "good" feature representation that can minimize the domain difference. It is applied to inductive and transductive learning.

  In [55], a convex optimization algorithm is proposed to learn the feature weights and meta-priors that are going to be transferred to different tasks whereas in [23], feature augmentation is used as a solution to the feature bias problem.

  In [73], the authors propose Transfer Component Analysis where common latent features with the same marginal probability distributions are learned by reproducing a kernel space. Then, traditional machine learning is applied, using the learned feature space as input. The Maximum Mean Discrepancy (MMD) is used to measure the differences between probability distributions.

  In [35] the authors propose a Stacked Denoising Autoencoder to solve the difference between marginal distributions in source and target domains. The input space is transformed to discover a common invariant latent feature space, and then a classifier is trained on the source domain using the new set of features and the target domain used for predictions.

- **Parameter transfer:** Works under the assumption that individual models for related tasks share parameters or a combination of hyper-

parameters. It is mostly applied to inductive and transductive transfer learning.

Most of the work related to parameter transfer referenced in [74] is within the scope of regularization and Bayesian theory. In [28] and [34], higher weights to the loss function in the target domain can lead to performance improvements.

In addition, parameter transfer algorithms work well in multi-task learning tasks, where a classifier learns from source and target domains at the same time [94] and [95]. It is worth mentioning that in these works *negative transfer* effect is mitigated by increasing the weighting of the instances that makes source and target domains more related, thus making the algorithm more robust to the addition of samples that are unrelated to the source domain.

- **Relational knowledge transfer:** Is applied to problems where there is some kind of relation in the data (e.g. network or social network data). It is mostly applied to inductive transfer learning.

  First-order and Second-order Markov Logic Networks are used in [65] and [24], respectively. Assuming that source and target domains are related, the proposed algorithms find a way of connecting entities and relationships between the source and the target domains.

The solution proposed in this thesis fits in the parameter transfer category since the neural network architectures and parameters used to solve source and target learning tasks are the same.

Remark 1. *The work carried out in this thesis fits in the transductive learning and parameter transfer categories.*

2.2 Deep Transfer Learning

Transfer learning has been applied with great success, improving the state of the art in a wide range of fields such as CV, NLP and Speech Recognition. Due to recent advances in deep learning, new approaches combine deep and transfer learning.

Three major approaches are identified when using deep neural networks and transfer learning together, in particular Convolutional Neural Networks (CNNs) for CV:

- **Fixed feature extractor**: Takes a pretrained CNN and removes the last layer. This pretrained CNN is used as feature extractor and a linear classifier is trained on the new dataset.

- **Fine-tuning**: Not only is the classifier on top of the new CNN trained, but also the weights on the pretrained layers are retrained through back propagation.
- **Pretrained models**: The pretrained CNN weights are shared at different checkpoints in time and used for fine-tuning on the new dataset.

These same techniques have been extended to other application fields such as NLP, Speech Recognition and Time Series Analysis where Long Short-Term Memory (LSTM) architectures are predominant. It is worth mentioning that, deciding the new neural network architecture, as well as finding a systematic approach to select the number of layers to fine-tune, remains an open research question. Current solutions are tailored to a particular domain. In the following sections, we discuss through examples, how deep transfer learning is applied in the different application domains.

2.2.1 Transfer Learning in Computer Vision

CV is one of the areas of application where transfer learning is widely used. The general approach is training a model on the new dataset by using a pretrained model on ImageNet [86] and fine-tuning a few layers. ImageNet is a largely annotated database used as benchmark for CV tasks. It allows researchers to train large models using Graphics Processing Units (GPUs) and share them with the community such that they can be used as a starting point for transfer learning.

As an example, in [69], mid-level image representations are learned on a large-scale annotated dataset. A CNN is trained on ImageNet and the learned image representations are transferred to different visual recognition tasks with limited amounts of training data. The authors significantly outperform the state of the art in object and action classification despite the difference in images statistics and tasks.

Motivated by the success of transfer learning in image classification, in [51], the authors leverage CNNs and transfer learning by training in a dataset of 1 million YouTube videos. They significantly improve the state of the art for video classification.

In [108], the authors analyze how transferable CNN features in CV are when training on ImageNet. Results show that, the transferability of features decreases when the distance between tasks increases, however transferring features, even from tasks that are distant, is better than random initialization. Moreover, they find a systematic way of quantifying the transferability of features from each layer in the neural network.

Transfer learning has also turned into the de-facto deep learning standard for medical imaging [82]. The authors use pretrained models on ImageNet for medical image classification. Unlike in the previous examples, here experiments show there are no significant benefits in terms of performance when using transfer learning in comparisons to more lightweight approaches. They propose weight transfusion and feature reuse in order to reduce convergence time as an alternative to transfer learning.

As noted before, even though more work on transfer learning has been carried out in the field of CV than in other fields, research questions such as how to fine-tune the new model remain open.

2.2.2 Transfer Learning in Natural Language Processing

Transfer learning has been successfully used in a variety of tasks in NLP such as general language understanding, question answering and natural language inference. The most successful approach so far is sequential transfer learning [85]. A model is pretrained on a large unlabeled text corpus and the resulting representations are adapted to a supervised target task .

Examples of pretrained models are: ELMo [78], Universal Language Model Fine-tuning (ULMFiT) [44], Bidirectional Encoder Representations from Transformers (BERT) [25] and Generative Pretrained Model (GPT-2) [81]. Before, the state of the art in NLP was dominated by simple two or three layer bidirectional LSTMs. Nowadays, the complexity has increased, for instance BERT and GPT-2 consist of 24 Transformer [98] blocks containing 1.5 billion of parameters.

The success of transfer learning in NLP is mainly due to the increasing model complexity of pretrained architectures and the large amount of training data used. For instance, in ULMFiT [44] the authors show that there are no big differences in performance when the amount of data used for the pretrained model is similar to the amount of data used for training a model from scratch. Similarly, Robustly-optimized BERT (RoBERTa) [60] outperforms BERT by training the same BERT model for a longer period of time.

Finding a systematic way of choosing the amount of network layers to fine-tune, as in CV, remains an open research question. In contrast, in NLP, fine-tunning layers progressively in time or intensity (i.e., at different learning rates) [44] has been shown to be successful improving the performance of the network on the new dataset.

Finally, some approaches look at reducing the complexity of the pretrained models. For instance in DistilBERT [87], the authors propose the

use of distillation techniques [41] to reduce the size of BERT without sacrificing much accuracy.

2.2.3 Transfer Learning in Time Series Analysis

Transfer learning has also been used for time series analysis, for classification and forecasting.

For instance, in [29] CNNs are used for time series classification. The authors propose Dynamic Time Warping (DTW) to measure the inter-dataset similarity and select the most appropriate source dataset. It is worth mentioning that, computing DTW is expensive since it is an Non-Deterministic Polynomial Time (NP)-hard problem, therefore heuristics are used. Due to this, in Chapters 5 and 6 instead, we propose the Euclidean Distance (ED). In our case we do not consider time delay since source and target time series should be as similar as possible for transfer learning.

Transfer learning for time series forecasting is proposed in: [83] for energy forecasting, [107] for financial time series and [63] for air quality prediction. As in NLP, this last work leverages pretrained LSTMs for transfer learning.

It is worth mentioning that, the application of transfer learning to time series analysis is not as widely explored as its application to CV and NLP. Transfer learning is a promising tool to solve the *cold start* problem [103] in time series forecasting. This happens when predictions need to be carried out for time series from which past observations are unavailable. In Chapters 5 and 6 of this thesis, we model the KPI prediction problem as a time series forecasting problem and use transfer learning to address current constraints in terms of data availability and time complexity of current approaches.

2.2.4 Transfer Learning and Reinforcement Learning

Another major trend is the joint application of transfer learning and Reinforcement Learning. Training a Reinforcement Learning agent can be infeasible due to data limitations, safety concerns, among other reasons.

For instance, in [75] training a self driving vehicle in a real life scenario would require non-affordable trials and errors. Therefore, the authors propose a translation network to make the model trained in a virtual environment workable in the real world.

2.3 Transfer Learning in Mobile Communication Networks

The applications of transfer learning in mobile communication networks are yet to be fully explored [15]. Two major areas of application are identified, with works focusing on either: (i) network optimization or (ii) network performance anticipation. In the sections below, we discuss relevant works in each area of application.

2.3.1 Transfer Learning for Network Optimization

The first major area of application includes studies focusing on the use of transfer learning for network policy optimization. This area of work is sometimes associated with reinforcement learning, where the main goal is as follows: given a set of possible short term decisions to take, find the optimum network policy in an autonomous manner such that the long term network performance improves.

Examples of this approach are the works in: [33] for network optimization, [7, 43] for caching, [111] for wireless networks design, [16] for resource management in Wireless Virtual Reality and [88] for resource allocation.

It is worth observing that one of the main motivations of this type of approach is decreasing the time complexity of algorithms when trying to find the best optimization policy. For instance in [88], the authors propose transfer learning for resource allocation as an alternative to Mixed Integer Nonlinear Programming (MINLP) problems, which are NP-hard.

Finally, other works focus on improving time complexity of evolving architectures in the 5G ecosystem. For instance, in [20] the authors propose transfer learning to create a collaborative system in the edge cloud for Multi-access Edge Computing (MEC), in which models at each edge can adapt to changes; obtaining significant improvements with regard to a more traditional centralized learning approach.

2.3.2 Transfer Learning for Network Performance Anticipation

In this second area of application, transfer learning is mostly used to anticipate a quantity of interest based on previous observations (e.g., user position, network KPI, etc.). Predicted values can be used by the operator to take an informed decision in order to optimize the network performance. This area of works applies supervised learning and makes different assumptions about the similarity between source and target tasks and domains.

In [70–72, 114], transfer learning is used for localization by transferring knowledge across devices, time and space.

Similarly, transfer learning has been used for outdoor position recovery in [113], traffic prediction in [112], fault classification in [100] and spectrum sensing in [77].

Our work is along the same line of the work carried out in [112], where deep learning is leveraged to anticipate a quantity of interest in the network. Network parameters are shared across architectures trained on different types of network traffic datasets. For instance, a network pretrained on the SMS dataset can be used to carry out predictions on a dataset containing calls traffic. In contrast, we utilize deep neural networks and select source and target domains according to domain distance metrics tailored to the kind of data. In Chapter 4, we also study the effect of data augmentation in the source domain when using transfer learning. In Chapters 5 and 6, we model the KPI prediction problem as a transfer learning time series forecasting problem and study the factors that affect the transfer learning error.

Remark 2. *The transfer learning approach proposed in this thesis utilizes deep learning and different domain similarity metrics which are tailored to the type of data, to address the existing challenges of data availability and complexity time of current anticipation approaches.*

CHAPTER 3

Deep Transfer Learning

In this chapter we introduce transfer learning and describe the main steps followed when applying transfer learning to different use cases in mobile communication networks.

3.1 Transfer Learning Background

A *domain* $\mathcal{D} := \{\mathcal{X}, P(X)\}$ consists of a feature space $\mathcal{X}$ and its probability distribution $P(X)$, $X \in \mathcal{X}$. A *task* $\mathcal{T} := \{\mathcal{Y}, f(\cdot)\}$ consists of a label space $\mathcal{Y}$ and a predictive function $f(\cdot)$, where $f(\cdot)$ can be written as $P(Y|X)$, $Y \in \mathcal{Y}$ and $X \in \mathcal{X}$. Formally, the definition of transfer learning is given as follows:

Definition 3.1.1 (Transfer Learning [74])**.** *Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.*

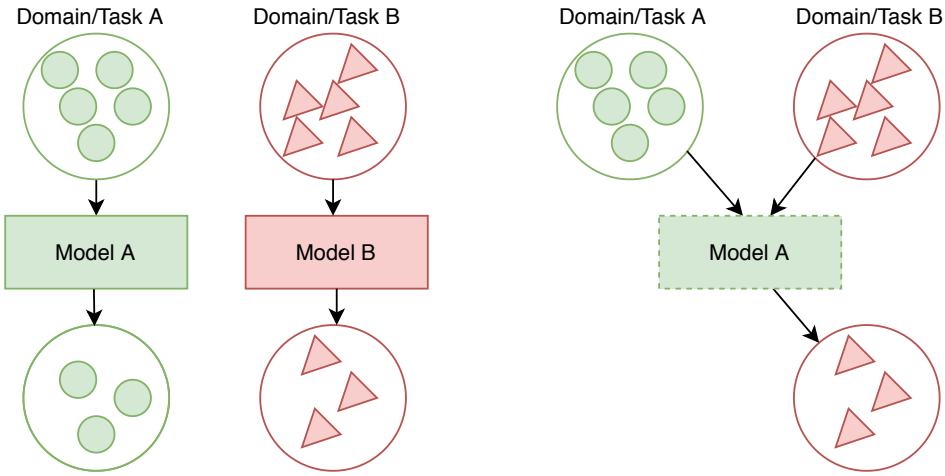


Figure 3.1: Traditional machine learning vs. Transfer learning

3.1.1 Traditional Machine Learning vs. Transfer Learning

Unlike transfer learning (see Definition 3.1.1), in traditional machine learning, source and target domains (i.e., $\mathcal{D}_S = \mathcal{D}_T$) and source and target tasks are the same, (i.e., $\mathcal{T}_S = \mathcal{T}_T$). Figures 3.1 shows the difference between both machine learning approaches.

In this thesis, we focus on cases where the source and learning tasks are the same (i.e., $\mathcal{T}_S = \mathcal{T}_T$), but source and target domain are "related" but different (i.e., $\mathcal{D}_S \neq \mathcal{D}_T$).

3.1.2 Transfer Learning Research Questions

According to [74], in a transfer learning problem, the main research questions to solve are:

1. *What to transfer?:* It refers to which part of the knowledge is going to be transferred across domains/tasks.

2. *How to transfer?:* It refers to the kind of algorithm that is going to be used for the transfer learning task.

3. *When to transfer?:* It refers to the situations in which transfer should be performed or not. It is highly related to avoid the *negative transfer* (i.e., hurting the performance of the transfer learning task on the target domain).

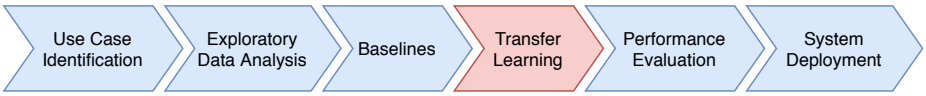


Figure 3.2: Transfer Learning pipeline

In this thesis, we model different use cases in the area of mobile wireless networks, as transfer learning problems. For each use case we provide answers to the research questions mentioned above.

3.2 Transfer Learning Pipeline

The main steps of our transfer learning pipeline are detailed in Figure 3.2. In the sections below we explain each step in detail.

3.2.1 Use Case Identification

Here, we identify potential use cases where the application of transfer learning could lead to improved performance in terms of accuracy, time complexity or data requirements. In all cases source and target domains are different but "related" and source and target tasks are the same. The selected use cases are summarized below:

- Tilt Dependent Radio-Map Predictions
- Anticipating Mobile Radio Networks KPIs with Transfer Learning
- Transfer Learning for Multi-Step Resource Utilization Prediction

3.2.2 Exploratory Data Analysis

We then carry out Exploratory Data Analysis (EDA) to gain the first insights from the data, evaluate its quality and studying the feasibility of using machine learning and specifically transfer learning. Examples of the techniques and tools used here are: boxplots, histograms and scatter plots as well as correlograms and hypothesis testing to assess stationarity in case of using time series data.

3.2.3 Baselines

Once the required data quality is achieved, we use statistics and traditional machine learning algorithms tailored to the type of data and the machine

learning problem to solve. Linear Regression [31], Random Forests [12] and k-nearest neighbors (k-NN) [19] are among the most popular algorithms used as benchmarks when working with categorical data in regression tasks (see Chapter 4). Similarly, Autorarima [11] and Moving Average [64] are popular for time series forecasting (see Chapter 5 and 6). These methods are used as benchmark to understand whether deep learning algorithm such as CNNs or LSTMs will have an impact on the prediction error and they are required.

In addition, we use a models with the same deep learning architecture used for transfer learning as benchmark. However, these models are either trained and tested on the available data from the target domain or trained on the source domain and applied to the target domain without the retraining step. In the next chapters, these two baselines are denoted with the suffixes S and BS, respectively. By using the S baselines, we can understand whether *negative transfer* is happening or not, since we compare the performance of transfer learning against the performance of the same method without transfer. By using the BS baselines, we can understand the benefits of using transfer learning (i.e., whether the retraining a model on the available data from the target domain improves the performance of the learning task in the target domain).

3.2.4 Transfer Learning

Firstly, we define source and target domains according domain distance metrics tailored to the type of the data. Finding the right domain distance metric for a given machine learning task is still an ongoing area of research. Proposed metrics in the literature vary depending on the dataset used for machine learning. For instance, the ED is used to measure the distance between two points in a n-dimensional space whereas the Cosine Similarity [97] can be used to measure the distance between documents or the Jaccard index [45] the distance between sets, just to mention a few. Similarly, metrics such as the Kullback-Leibler divergence index (KL), the MMD and the Jensen-Shannon divergence (JSD) can be used to measure the distance between data distributions [1]. In Chapter 4, we define our domain distance metric, namely the SD_{KL}, which is the symmetric version of the KL and it is used to measure the distance between the different probability distributions representing the different tilt-dependent radio maps. In Chapters 5 and 6, we use the ED, which is a common and efficient way of measuring similarity between time series in comparison to the DTW or the Pearson Correlation [97]. The main goal when using transfer learning is having

source and target domains as similar as possible.

Secondly, we carry out the transfer learning step by adapting traditional machine learning algorithms for the transfer learning task by freezing and retraining different parts of the system. Finding source and target domains as well as the neural network architecture that better fits the learning task, addresses the main transfer learning questions of *What, When and How to transfer?*. Common architectures adapted for the use of transfer learning are Feed-Forward Neural Networks (FFNs), CNNs and LSTMs. More details about each architecture and how they are adapted for transfer learning are shown in Chapters 5, 6 and 7, respectively.

3.2.5 Performance Evaluation

Finally, we carry out performance evaluation according to well known performance metrics such as Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE) tailored to the type of data. We also measure the training duration to evaluate the time complexity of each approach. If there are advantages in using a transfer learning, then the proposed algorithms can be deployed to production.

3.2.6 System Deployment

The system deployment phase comprises the implementation of the training, retraining and prediction functionalities as decoupled containerized microservices and their deployment as Representational State Transfer (REST) Application Programming Interfaces (APIs), endpoints which are suitable for integration with the rest of the Nokia portfolio.

CHAPTER 4

Applications to Tilt-Dependent Radio Map Prediction

In this chapter we propose a transfer learning framework for radio map reconstruction.

4.1 Motivation

Machine learning tools will be able to assist network operators in dealing with the increasing complexity of configuring parameters for network optimization by exploiting the increased availability of data in 5G coming from network devices and user terminals. Among the manifold parameters that can be configured at the base station (BS), one of the most important is the antenna tilt, which is the angle formed by the vertical direction which the antenna is facing and the horizon. Antenna tilt can be controlled either mechanically (by physically tilting the antenna up or down) or electronically (relying on beam-forming techniques that steer the main beam of the antenna towards a desired vertical direction), or by a combination of the two. The antenna tilt directly impacts the performance of the cell served by the BS in terms of network coverage, signal strength and inter-cell interference,

and therefore determines the quality of service experienced by end users. In particular, when the antenna tilt is changed in the BS, its effect on the antenna gain over distance also changes, which further leads to a change of the Reference Signals Received Power (RSRP) values [22]. Therefore, different radio maps can be generated as a function of the selected tilt configuration. We refer to these as *tilt-dependent radio maps*.

From an operator's perspective being able to predict cell performance without carrying out extensive trials or measurement campaigns is of key importance for two reasons: firstly, extensive measurement campaigns, such as test driving, are time consuming and costly. Secondly, even if measurements were obtained inexpensively (e.g., directly from user terminals through crowd-sourcing), testing all possible antenna configurations might still be impractical at network runtime. Given such difficulties, a solution which is particularly appealing to network operators is transferring the knowledge acquired from a single measurement campaign (for a given antenna tilt setting) to a new *domain* (a new tilt setting) without needing to acquire a complete set of additional measurements. Therefore, in this chapter, we formalize and solve this problem via *transfer learning*, a paradigm that has received increasing attention in the last few years [74].

4.2 Related Work

Tilt-dependent radio map prediction plays a crucial role in the context of network planning and proactive network optimization [66]. The predicted propagation condition can be exploited for a reliable decision making process to dynamically optimize antenna tilts in a time-varying network environment [66, 68]. Although radio map prediction has been extensively studied [79], its dependency on antenna tilt has been investigated only in few works. The authors in [84] propose a geometrical-based extension to various traditional log-distance path loss models (Okumura-Hata, Walfisch-Ikegami) to take into account the antenna tilt during the prediction of the signal strength at a given distance from the BS. The proposed extension, named vertical gain correction (VGC), is calculated directly from the antenna patterns provided by the manufacturer and is added to the signal strength estimated by the path loss models to compensate for the antenna tilt. Experimental results on data collected from LTE BSs show that VGC improves the performance of signal strength prediction when compared to traditional models. Similarly, the work in [52] investigates the effect of antenna tilt on radio maps, comparing the path loss models developed by the 3rd generation partnership project (3GPP) [2] for different propagation

environments. The results were obtained using a ray tracing tool able to take into account antenna tilts and demonstrate that changing antenna tilt has a significant impact on the shadowing map. This calls for a rethinking of currently available 3GPP propagation models and assumptions, which apply an identical shadowing map independently from the antenna tilt.

4.3 Problem Statement

We address the following problem: "*how to predict the performance of a given network configuration by leveraging information from different network configurations*." The performance measure that we target is the received signal strength in the downlink. The network configuration domains include the tilting configurations of the transmitting BSs.

We consider a BS that can work in H different tilt configurations, indexed by $h = 1, \ldots, H$. Let $s_h(\mathbf{x}_i)$ be the measured signal strength received at location $\mathbf{x}_i = \{y_i, z_i\}$ when the h-th tilt configuration is selected at the BS, where y_i and z_i indicate the latitude and the longitude of the i-th location, respectively. Let $\mathcal{M}_h$ be the set of location indexes where measurements have been taken with configuration h.

The problem at hand can be defined as follows: given $\{s_h(\mathbf{x}_i) : i \in \mathcal{M}_h\}$, estimate the unknown signal strength $\hat{s}_n(\mathbf{x}_j)$ at the same or different locations, $\mathbf{x}_j$, with $j \in \mathcal{M}_n$, under different network configuration domains, $n \neq h$.

4.4 Data Collection

The dataset used in this work is composed of reference signal received power (RSRP) outdoor measurements collected in Espoo, Finland, in November 2016 from two commercial LTE BSs with three different 120° sectors each and operating at 2.6 GHz. Figure 4.1 shows the positions of the two antennas and the representation of the target area. The measurements were collected from three different physical cell identifiers (PCIs), which will be referred to as PCI 1, 2 and 3. PCIs 1 and 2 refer to two different sectors of the same BSs, whereas PCI 3 is a sector of a different BS.

RSRP measurements were collected using an Android device equipped with an application capable of storing the RSRP from all the received cells, the cell identifier, the global positioning system (GPS) position of the device and the timestamp. Such measurements were carried out at a frequency of 1 Hz while walking along routes of 8 km within each cell coverage area, with a minimum and maximum distance from the BS of 30 m and 900 m,

Figure 4.1: Map showing the BS positions and the PCIs in the reference dataset

respectively. By design, the testing paths were planned to include different propagation conditions: university campus with two or three-story buildings, residential areas, parking lots, lower density rural and open areas with different types of roads (e.g., pedestrian, cycling and main roads). Each testing path was walked once for each electronic tilt setting. The available tilt settings are 2, 3 and 6 degrees for each PCI, respectively. The receiver was placed at the height of 1.5 m and always kept at the same orientation. The weather conditions were stable and cloudy, and the route was covered by snow for most of the measurement campaign. The RSRP values were collected from an operating mobile network. According to [4], these values include the power from co-channel serving and non-serving cells as well as adjacent channel interference, but only on the resource elements that carry reference signals. Since these values are measured only in the symbols carrying the reference signal, they exclude most of the wide band noise and interference from other cells. Overall, they are proportional to the SNR on average [5]. Therefore, they are still a good indicator to be used in radio map reconstruction, reflecting the channel propagation conditions.

4.4.1 Data Preprocessing

In total, about $3 \cdot 10^5$ RSRP measurements were obtained. Each observation contains the following fields:

- Measurement position (latitude and longitude coordinates)
- RSRP value (downlink signal strength)
- PCI (physical cell identifier)

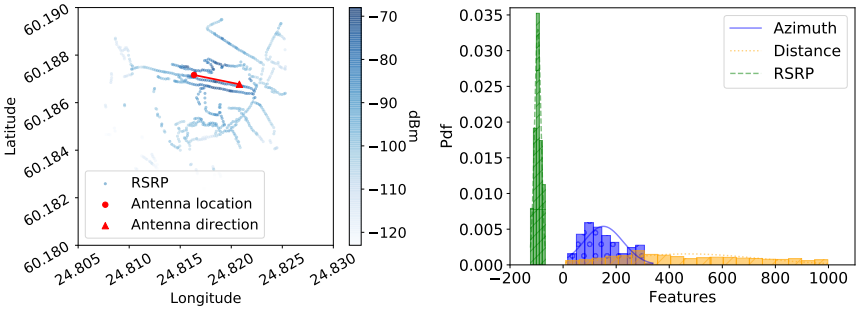
The raw dataset was preprocessed to remove corrupted samples: for example, at the beginning of each experiment the GPS receiver requires some initialization time during which position is recorded incorrectly. Moreover, we overlaid the considered area with a grid. For each grid element of size 20 m $\times$ 20 m, we replaced the RSRP values with their average to reduce noise. After the preprocessing steps, the reduced dataset consisted of ~600 observations per PCI and per tilt configuration, for a total of ~$5.8 \cdot 10^3$ measurements. Before training, the data is scaled between 0 and 1 by using a Min-Max scaler, which is fitted to the training set and applied to the cross validation and test sets. The scaling transformation is then reversed before evaluating the algorithm performance. In our previous work [76], we analyze the transferability across different tilt settings of the same PCI as well as the transferability across different PCIs. In particular, we show that the transferability within the same PCI is much higher than the transferability across different PCIs. Therefore, we focus hereafter on the task of transferring the knowledge from one tilt configuration to another within the same PCI. Unlike our previous work, in which the standard machine learning tools are used, we apply transfer learning with deep neural networks.

Figure 4.2 shows a representation of the data collected for different tilt configurations of PCI 1. Figures 4.2a, 4.2c, 4.2e show RSRP values (in dBm) over the considered geographic area, when the antenna was tilted at 2, 3 and 6 degrees, respectively. It can be observed that the spatial distribution of the data follows a similar pattern for different tilt configurations of PCI 1. For example, points located in the main direction of the antenna have higher signal strength values than the rest of the points. In addition, points closer to the antenna also follow a similar pattern, while points which are far apart have lower RSRP values. To give an idea of the domain differences, in Figures 4.2b, 4.2d, 4.2f we show the normalized histograms, and the continuous approximations of the PDFs of the azimuth, distance and RSRP for the different tilt configurations of PCI 1.

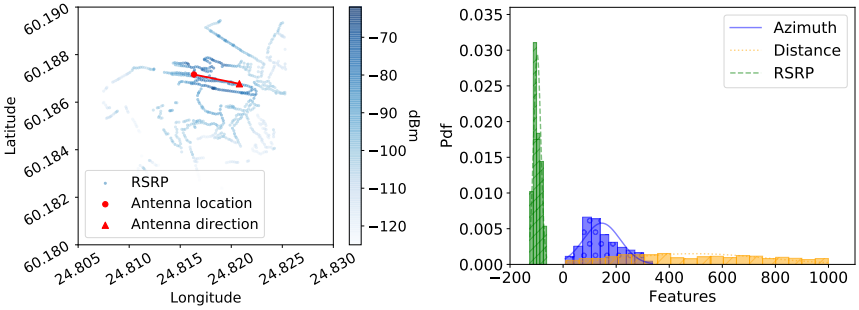
Even if some similarities can be observed between the statistical characteristics of the data collected under different tilt configurations, the data does not come from the same distributions. For example, the azimuth distribution for a greater tilt value (Figure 4.2f) has a lower standard deviation than the distributions for lower tilt values (Figures 4.2b and 4.2d).

4.5 Prediction Approaches

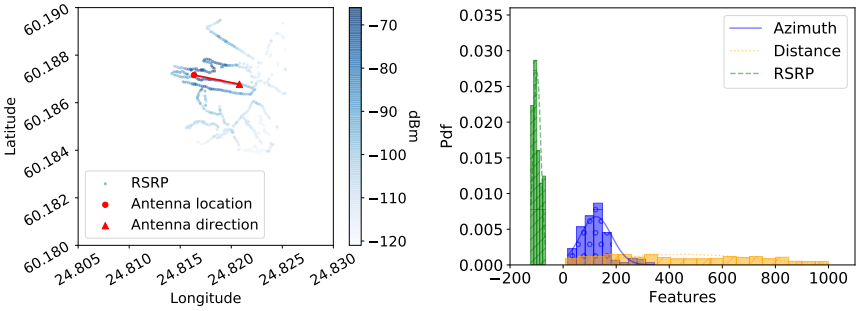
Given the base station location $\mathbf{x}_A$, let $\mathbf{x}$ and h be the target position and the configured antenna tilt, respectively. The following set of features, derived



(a) RSRP values at the sampled points, Tilt 2 (b) probability density function (PDF), Tilt 2

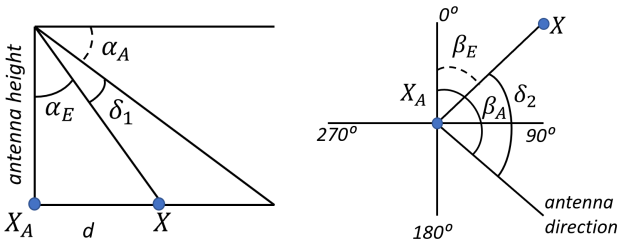


(c) RSRP values at the sampled points, Tilt 3 (d) PDF, Tilt 3



(e) RSRP values at the sampled points, Tilt 6 (f) PDF, Tilt 6

Figure 4.2: Tilt-dependent radio maps, normalized histograms and PDFs for three metrics RSRP, azimuth and distance, PCI 1



(a) Relative distance (b) Relative azimuth

Figure 4.3: Relative angles on the vertical (left) and horizontal (right) planes between the antenna pointing direction and the direction towards the test position $\mathbf{x}$

from $(\mathbf{x}, h)$ and shown in Figure 4.3, is considered for the prediction task:

- the physical *distance* between the antenna and the measurement position, $d(\mathbf{x}) := d(\mathbf{x}, \mathbf{x}_A)$

- the *relative elevation angle* between the down-tilt of the antenna and the vertical direction from the antenna emitting element to the measurement position, defined as:

$$\begin{aligned} \delta_1(h, \mathbf{x}) &= 90° - (\alpha_A + \alpha_E(\mathbf{x}, \mathbf{x}_A)) \\ &= 90° - (h + \alpha_E(\mathbf{x}, \mathbf{x}_A)), \end{aligned} \tag{4.1}$$

  where $\alpha_A := h$ is the antenna down-tilt (mechanical plus electrical) and α_E is the angle at which the antenna 'sees' the target position depending on the antenna position $\mathbf{x}_A$ and the target location $\mathbf{x}$

- the *relative azimuth* between the horizontal orientation of the antenna and the horizontal direction to the measurement position defined as:

$$\delta_2(\mathbf{x}) = \beta_A - \beta_E(\mathbf{x}, \mathbf{x}_A), \tag{4.2}$$

  where β_A denotes the horizontal orientation of the antenna and β_E is the horizontal orientation of the target position with respect to the antenna position

Each sample in the training dataset is, therefore, associated with a tuple of values (d, δ_1, δ_2) . The logarithmic transformation is applied to d since the RSRP values are measured in dBm. Finally the feature vector $[d(\mathbf{x}), \delta_1(\mathbf{x}, h), \delta_2(\mathbf{x})]^T$ is obtained and used as input to our models.

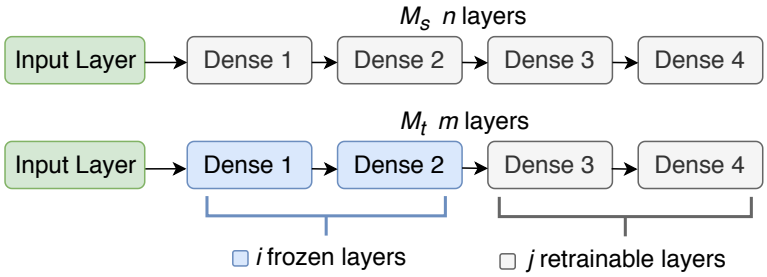


Figure 4.4: Transfer learning model

4.5.1 Transfer Learning Approach

The proposed transfer learning approach has been inspired by the fields of computer vision and natural language processing [69] [18], where deep neural networks constitute the state of the art for classification and prediction tasks. The core idea of our approach is to train a neural network for the signal strength prediction task in a source domain (reference tilt configuration) and then build a new neural network to obtain fine-grained predictions in the target domain (target tilt configuration). More details on how both neural network architectures and parameters are obtained are shown in Section 4.6.2. The neural network architectures used in our approach are FFNs, which are well-known for being powerful nonlinear function approximators [36]. We opt for FFNs instead of more complex network architectures, such as CNNs or Recurrent Neural Networks (RNNs) for two main reasons. Firstly, from preliminary experimental results (see Figure 4.6), we observed that the achieved training and cross validation losses are already very low and close to each other for the problem at hand. Therefore, using a more complex architecture with the same limited amount of data available for training could lead to a bigger gap between training and cross validation, causing overfitting and thus worsening the performance. Secondly, more complex architectures would require more parameters and hyperparameters to be found, causing an increased training time.

We use the mean square error (MSE) as loss function, which is the standard metric used in regression tasks. Here the goal is to minimize the difference between the real and predicted RSRP values. It is worth noting that the MSE is well known for being sensitive to outliers, however this is not a concern in this case since outliers have been removed in previous preprocessing steps (see Section 4.4.1). By using FFNs as the basic building blocks of our architecture, the flow of information only travels forward, and the layers of the network are fully connected. Formally, FFNs learns a

combination of parameters to find the best function approximation. In our case, we aim at finding a set of parameters $\boldsymbol{\theta}$ for the hidden layers and a set of parameters $\mathbf{w}$ for the output layer to estimate $\hat{s}(\mathbf{x}) \in \mathbb{R}^q$ for $\mathbf{x} \in \mathbb{R}^p$, as shown in Eq. (4.3):

$$\hat{s}(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}, \mathbf{w}) = \boldsymbol{\phi}(\mathbf{x}, \boldsymbol{\theta})^T \mathbf{w} \tag{4.3}$$

where $\boldsymbol{\phi} : \mathbb{R}^p \rightarrow \mathbb{R}^q$, is a nonlinear transformation defining the hidden layers, and parameters $\mathbf{w} \in \mathbb{R}^q$ map from $\boldsymbol{\phi}$ to the desired output. Each input is represented by a tuple containing distance, relative azimuth and relative angle (i.e., (d, δ_1, δ_2)) and each output is the RSRP value $\hat{s}(\mathbf{x})$ associated to a given input. Therefore, $p = 3$ and $q = 1$ are the input and output dimensions, respectively.

The proposed transfer learning approach is composed of the following:

- $\mathcal{D}_S$: source domain which consists of the feature space of the reference tilt configuration and its marginal probability distribution
- $\mathcal{D}_T$: target domain which consists of the feature space of the target tilt configuration and its marginal probability distribution
- $\mathbf{M}_S = \hat{f}_S(\cdot)$: an FFN with n layers approximating the predictive function in the source domain $f_S(\cdot)$
- $\mathbf{M}_T = \hat{f}_T(\cdot)$: an FFN with m layers approximating the predictive function in the target domain $f_T(\cdot)$
- $\{p_1, ..., p_K\}$: the best combination of hyperparameters shared by both FFNs associated with the source and target domains respectively[1].

The steps of our transfer learning algorithm are defined as follows:

1. We select the source domain $\mathcal{D}_S$ and train $\mathbf{M}_S$ on $\mathcal{D}_S$, finding the best combination of hyperparameters $\{p_1, ..., p_K\}$. We use Bayesian optimization [90] since it is an effective way of finding a suboptimal solution in less time, when compared to random search [8], for example. The problem of choosing the hyperparameters is modeled as a sample of a Gaussian process (GP). We start with an initial combination of hyperparameters and dynamically update the searching space based on the built surrogate probability model mapping from hyperparameters to the probability of a score on the objective function (see

[1]For parameter transfer we assume that the models for source and target domains share a combination of hyperparameters.

Section 4.6.2 for numerical results). It is worth noting that the optimization process has been carried out on $\mathcal{D}_S$ since we assume we do not have enough data available from $\mathcal{D}_T$ to find a model that perform well on $\mathcal{D}_T$. Moreover, the main goal of our approach is learning the best model possible on $\mathcal{D}_S$ by using sufficient data and transferring this knowledge to $\mathcal{D}_T$ which has limited data. After the Bayesian optimization step, $\mathbf{M}_S$ is trained on $\mathcal{D}_S$.

2. Once we have obtained $\mathbf{M}_S$, we model $\mathbf{M}_T$ by taking the first $i \leq m$ layers of $\mathbf{M}_S$ with the associated weights and adding new $j \leq n$ layers that are initialized with random weights. The reason for this is that the first layers of the network can capture more general characteristics about the feature space, while the latter ones capture more specific behaviors. For choosing the best values of i and j, we tried all possible combinations of values such that $0 \leq i \leq m$ and $j = m - i$ and selected the one that led to the best accuracy (details are provided in Section 4.6.2). Figure 4.4 shows a graphical representation of $\mathbf{M}_S$ and $\mathbf{M}_T$, where $\mathbf{M}_T$ contains the first three layers of $\mathbf{M}_S$ and two new layers.

3. Finally, we train $\mathbf{M}_T$ on the few data available from $\mathcal{D}_T$ using the hyperparameters $\{p_1, ..., p_K\}$. We freeze the first i layers and retrain only the last j layers of $\mathbf{M}_T$ with data from $\mathcal{D}_T$. We refer to this approach as DNN T. $\mathbf{M}_S$ and $\mathbf{M}_T$ have the same complexity (i.e., number of layers and hidden units) in order to ensure fairness when comparing $\mathbf{M}_S$ and $\mathbf{M}_T$. In addition, using a much more complex architecture with a limited amount of data on $\mathcal{D}_T$ is more likely to increase overfitting and worsen the performance, while using a much simpler architecture does not improve performance (see DNN T 2F 1R on Figure 4.7).

We use the Keras framework [17] on top of TensorFlow [3] due to its flexibility for implementing this transfer learning approach and performing hyperparameters search. In total, the training and testing phases of the two models do not last more than two minutes. We use a laptop with 16 GB of RAM and a 7th generation, Intel Core i7 processor.

4.5.2 Baseline Methods

In this section, we describe the baseline methods used to benchmark our work: heuristic (H) using data provided by antenna manufacturer as well

as k-NN and Random forest (RF), which performed the best for the task at hand in our previous work [76].

Heuristic

This is the simplest baseline method, where the predicted values are extracted from the data sheets provided by the antenna manufacturer. Given a set of locations at a given tilt configuration, for each sample we create the feature vector by calculating *distance*, *relative angle* and *relative azimuth* (Section 4.5). In a second step, we use the data sheets provided by the antenna manufacturer to extract the antenna gain on the vertical and horizontal planes. Finally, we apply the path loss model to calculate the predicted values. Formally, the process is defined as follows:

1. Given $\mathcal{M}_h$ as the set of location indexes where measurements for the considered base station running configuration h have been taken, we calculate for each location $\mathbf{x} \in \mathcal{M}_h$ a tuple of values (d, δ_1, δ_2). Then we create the feature vector $\mathbf{z} := [d(\mathbf{x}), \delta_1(\mathbf{x}, h), \delta_2(\mathbf{x})]^T$ as shown in Section 4.5.

2. Let $\eta(\mathbf{x})$ and $\gamma(\mathbf{x})$ be the horizontal and vertical gain of the antenna in dB, respectively, as taken from the manufacturer antenna sheets. Given the known position $\mathbf{x}$, we formally define $\Delta(\mathbf{x})$ as:

$$\Delta(\mathbf{x}) = \eta(\mathbf{x}) + \gamma(\mathbf{x}) \tag{4.4}$$

3. Given $\Delta(\mathbf{x})$, we use the path loss model to generate the labels, $\hat{s}(\mathbf{x})$, by applying the following:

$$\hat{s}(\mathbf{x}) = \phi_0 + \phi_1 10 \log(d(\mathbf{x})) - \Delta(\mathbf{x}), \tag{4.5}$$

   where ϕ_0 and ϕ_1, similar to [84], are the linear regression coefficients calculated for the reference dataset.

k-Nearest Neighbors with Inverse Distance Weighting

This technique is one of the simplest multivariate interpolation methods which extends the classical nearest neighbor approach [19]. We apply the technique on the same feature vector $\mathbf{z}(\mathbf{x}) := [d(\mathbf{x}), \delta_1(\mathbf{x}), \delta_2(\mathbf{x})]^T$ as defined in the above-mentioned Heuristic approach. It predicts the signal at an unknown target location $\mathbf{x}$ (corresponding to a feature vector $\mathbf{z}(\mathbf{x})$) as a

weighted average of the signals at the k locations with the closest distances calculated based on feature vectors.

$$\hat{s}(\mathbf{z}) = \sum_{i \in \mathcal{M}(\mathbf{z})} \omega_i s(\mathbf{z}_i) \tag{4.6}$$

The set $\mathcal{M}(\mathbf{z})$ includes the feature vectors which are the closest to the unknown target vector $\mathbf{z}$, with cardinality $|\mathcal{M}| = k$. Weights ω_i are chosen to be inversely proportional to the distance $d(\mathbf{z}_i, \mathbf{z})$ and their sum is normalized to one, using the equation below:

$$\omega_i = \frac{d(\mathbf{z}_i, \mathbf{z})^{-1}}{\sum_{j \in \mathcal{M}(\mathbf{z})} d(\mathbf{z}_j, \mathbf{z})^{-1}}. \tag{4.7}$$

Random Forest

Is one of the ensemble methods used for classification and regression purposes. The algorithm was introduced by Ho [42] in 1995, and later extended by Breiman and Cutler [58], it uses the idea of bagging to perform predictions. During the process several trees are grown independently using different bootstrapped samples of the data and majority voting or averaging is used for the final prediction. In contrast to traditional trees, the variable used to perform the split in each node is chosen randomly from a set of predictors [58]. RF is known to sometimes outperform other machine learning techniques, such as neural networks, due to its resistance to overfitting [12].

4.6 Experimental Setup

In this section, we describe the set of experiments carried out. We compare the prediction error of our transfer learning method (i.e., DNN T in Section 4.5.1) against the baseline methods (i.e., H, k-NN, RF in Section 6.4.1). In the following, the suffix T is used to denote the transfer learning approach (i.e., DNN T). Similarly, the suffix S is used to denote the methods that do not use transfer learning (i.e., H S, k-NN S, RF S and DNN S). It is worth noting that DNN T is trained on data from a different tilt configuration (source domain) whereas H S, k-NN S, RF S and DNN S are trained on data from the same tilt configuration (target domain). In this way, we compare the performance of the proposed transfer learning solution against the performance of traditional machine learning solutions to reveal the scenarios where a transfer learning solution is preferred. We also train a model on the source domain and apply it to the target domain without the retraining and fine tuning step. This last approach is referred as DNN BS. It does not

require data from the target domain since no retraining is performed. In this case, the purpose is carrying out comparisons against the transfer learning solution to evaluate the real need for the retraining and fine tuning step.

We carry out two different sets of experiments that differ in the way the source domain is built. In Section 4.7.1, the source domain consists of measurements from a single tilt configuration, which differs to the one used for target domain. In Section 4.7.2 we augment the source domain by adding measurements from other available tilt configurations of the same PCI. In both cases, we analyze the impact on the performance when a limited amount of data from the target domain is available in the training phase. We study two strategies to select data from the target domain: (i) uniformly distributed in the reference area or ii) non-uniformly distributed according to a predefined sampling strategy (i.e., different distance ranges from the antenna location).

For each tilt configuration the amount of data available is about 600 measurements. In all the experiments, the data is divided into training, cross validation and test sets. We use 80% of samples for training, 10% for cross validation and 10% for testing. We vary the quantity of data taken from the target domain for training or fine tuning. For the DNN T, this is the number of samples used to train and fine tune $\mathbf{M}_T$. For the k-NN S, RF S and DNN S this is the quantity of data available for training a model on the target domain using data from the same target domain. In contrast, H S does not need training data. One of the main objectives is to map the amount of labeled data required from the target domain and corresponding performance, assessed in terms of MAPE, which is defined as:

$$\mathrm{MAPE} = \frac{100}{k} \sum_{i=0}^{k-1} \left| \frac{s_i - \hat{s}_i}{s_i} \right|, \tag{4.8}$$

where k is the number of target positions in the testing dataset.

4.6.1 Domain Distance

Since the performance of the transfer learning approach depends on the similarity between the training and testing sets on the target domain, we introduce a measure of the *degree of similarity* between datasets which is then used throughout this section. We quantify similarity in terms of KL divergence index [48], which measures the relative entropy of a given probability distribution with respect to another one. Given two reference datasets, one used for training and one used for testing (both in the target domain), we

derive the KL divergence indexes of the probability distributions of the logarithm of the distance (d), relative angle (δ_1) and relative azimuth (δ_2) of the two datasets. Formally, the symmetric KL divergence index of the distance probability distributions is given by:

$$\begin{aligned} SD_{KL}(d) = &\sum_{i=1}^{k} P_d^{(\text{tr})}(i) \log \frac{P_d^{(\text{tr})}(i)}{P_d^{(\text{te})}(i)} \\ &+ \sum_{i=1}^{k} P_d^{(\text{te})}(i) \log \frac{P_d^{(\text{te})}(i)}{P_d^{(\text{tr})}(i)}, \end{aligned} \tag{4.9}$$

where $P_d^{(\text{tr})}(i)$ and $P_d^{(\text{te})}(i)$ with $i = 1 \dots k$ define the discrete probability distributions of the distance in the training and testing sets of the target domain, respectively and k is the amount of bins used to estimate either $P_d^{(\text{tr})}$ or $P_d^{(\text{te})}$. Similar definitions hold for the KL divergence indexes related to the relative angle δ_1 and relative azimuth δ_2. Finally, to give a more succinct representation of domain similarity, we introduce the domain distance (DD) measure by summing the three indexes together:

$$\text{DD} = SD_{KL}(d) + SD_{KL}(\delta_1) + SD_{KL}(\delta_2). \tag{4.10}$$

Figure 4.5 shows the average DD across PCIs for all the possible combinations of training and testing sets on the target domains and the amount of points from the target domain used for training. The solid curve in Figure 4.5 shows the DD when the available samples taken from the target domain for training or fine tuning are uniformly sampled in the reference area. The dashed curve in Figure 4.5 shows the DD when the samples are taken between 300 m and 600 m of distance from the antenna location. Smaller DD values indicate higher domain similarity and vice versa. For instance, when the samples are uniformly distributed the similarity between training and testing sets in the target domain is higher, which makes the DD values lower, i.e., they range from 0.8 to 1.3 (see Figure 4.5 solid curve). In contrast, when the available measurements are located at a certain distance range from the antenna (i.e., 300 to 600 m), similarity is lower, meaning the DD values are higher ranging from 1.5 to more that 2 (see Figure 4.5 dashed curve).

4.6.2 Parameters Optimization

Network parameters are chosen in a hybrid manner by using a mixture of Bayesian optimization and manual fine tuning.

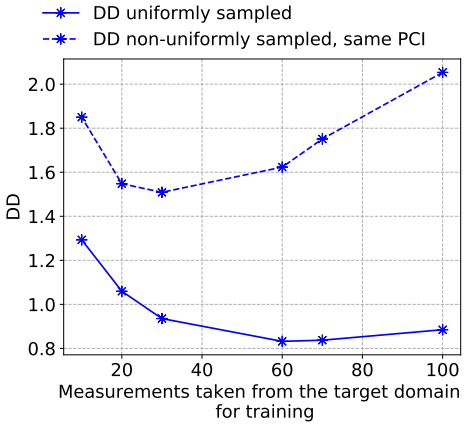


Figure 4.5: Domain distance

Parameters search on $\mathcal{D}_S$

Bayesian optimization requires, as starting point, a network architecture that converges. It also requires that the hyperparameters search space is specified. We begin with an architecture that contains 4 layers with $4, 10, 4$ and 1 hidden units, respectively, $1e-3$ as the initial learning rate and ReLu as the initial activation function. The activation function search space contains Sigmoid and ReLu, and the learning rate search space goes from $1e-7$ to $1e-1$. After 50 iterations the process converges and we find out that for all the tilt configurations and PCIs the best learning rate is approximately 0.099, and the choice of activation function that leads to the minimal error is Sigmoid. During this process, the model is shown to achieve good performance in the source domain. Figure 4.6 shows the training and cross validation errors for PCI 1 and tilts 2, 3 and 6. Comparable results are achieved for the rest of the PCIs and tested tilt combinations. It can be observed that the training and cross validation errors decrease dramatically during the first 150 epochs for tilts 2 and 3 and the first 20 epochs for tilt 6. After this they keep decreasing steadily, becoming very close to 0, which means the chosen architecture fits the data coming from $\mathcal{D}_S$. We note that both errors are close to each other, meaning that our model is not overfitting. Once the learning rate and activation functions are chosen, we carry out experiments on $\mathcal{D}_S$, increasing and decreasing the model complexity by adding and removing layers and hidden units, respectively. Table 4.1 shows the MAPE obtained leveraging the tested architectures. The reported MAPE is an average across all the available PCIs and all the available combinations of tilt configurations as source and target domain. It can be observed that

$\mathcal{D}_S$ Network architecture	Avg. across Tilts
4, 10, 4, 1	**4.30**
4, 10, 4, 4, 1	11.94
4, 10, 1	5.58
4, 20, 8, 1	6.82
4, 5, 2, 1	5.59

Table 4.1: MAPE for variations in the amount of layers and hidden units for PCI 1, Tilt 2, 3 and 6

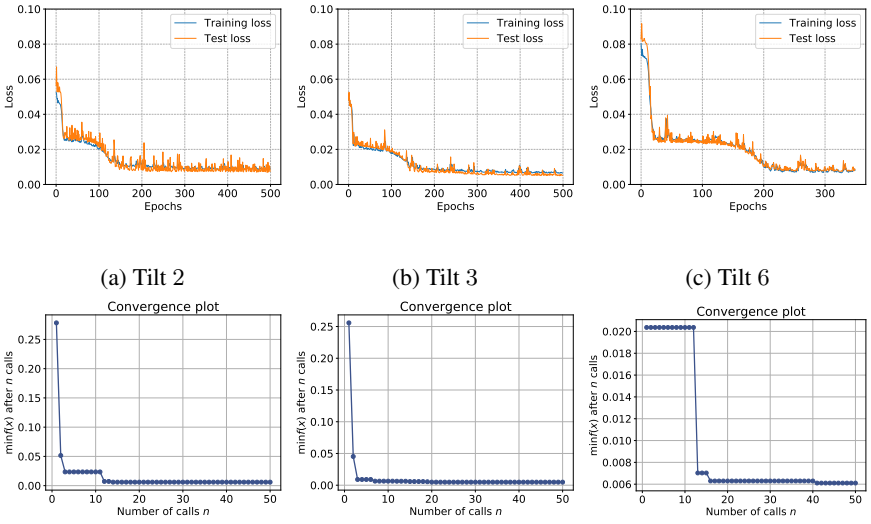
Number of epochs	500
Batch size	128
Number of inputs	3
Number of layers	4
Hidden units per layer	4, 10, 4, 1
Activation function	Sigmoid
Optimizer	Adam
Learning rate	0.099

Table 4.2: Hyperparameters found by Bayesian and manual optimization

increasing or decreasing complexity worsens the performance for all the tilt configurations on average, therefore the initial combination of 4 layers containing $4, 10, 4$ and 1 hidden units respectively is the one that leads to the best performance. Table 4.2 summarizes the best combination of hyperparameters found by a mixture of Bayesian and manual optimization.

Frozen and re-trainable layers

As explained in Section 4.5.1, the amount of layers to freeze and retrain is chosen through an empirical approach, trying all possible combinations and choosing the best one. Figure 4.7 reports the MAPE averaged across all the PCIs and all the possible combinations of tilt configurations as source and target domain when using different numbers of layers to freeze, i and the number of retrainable layers, j. DNN T method indicates that the weights in the retrainable layers of $\mathcal{M}_T$ are randomly initialized, whereas the DNN T W indicates that the weights in the retrainable layers are initialized with the weights from $\mathcal{M}_S$ after training. We use F and R to denote the number of layers to freeze and retrain on $\mathcal{M}_T$, respectively. We select $i = 2$ and $j = 2$ (i.e., DNN T 2F 2R), since it is the combination of values that leads to the best MAPE on $\mathcal{D}_T$.



(a) Tilt 2 (b) Tilt 3 (c) Tilt 6

(d) Tilt 2 (e) Tilt 3 (f) Tilt 6

Figure 4.6: Training curves and Bayesian convergence on the source domain, PCI 1

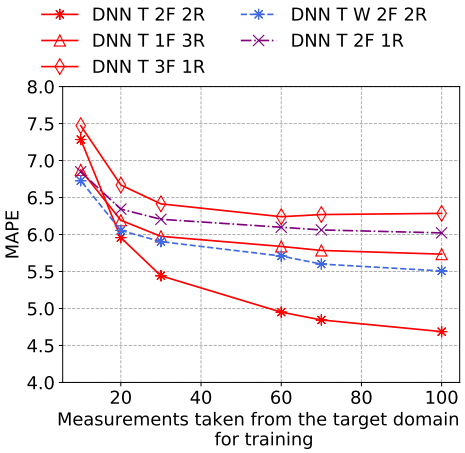


Figure 4.7: MAPE for the different values of frozen (F) and retrainable (R) layers using random or source domain weights initialization

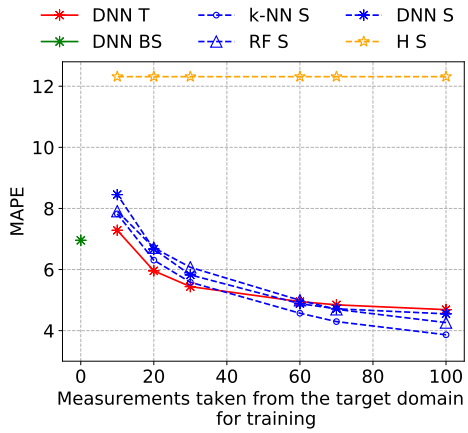


Figure 4.8: MAPE when training or fine tuning on uniformly sampled measurements

4.7 Results

4.7.1 Single Tilt Transfer

We use a dataset obtained under a given tilt setting (source domain) to predict the performance of the same antenna under a different tilt configuration (target domain). In particular, we consider two different scenarios: when the data available from the target domain is limited and sampled uniformly (see Section 4.7.1) and when the data is still limited but sampled according certain criteria, for instance at a given range of the antenna location (see Section refsssec:lim$_n on_u niform$).

Limited and uniformly sampled measurements

In this case measurements represent a wide range of relative distances, azimuth and RSRP values. The amount of instances taken for training or fine tuning varies between 0 to 100. Figure 4.8 shows the average MAPE across all the PCIs and possible pairs of training and testing tilt combinations, obtained by the different prediction approaches described in Section 4.5. We can draw the following conclusions:

- All the machine learning methods (i.e. k-NN S, RF S, DNN S, DNN T) outperform the heuristic approach (i.e. H S) for any number of instances taken from the target domain for training or fine tuning. Therefore, the machine learning algorithms trained on real data are more effective at capturing the non linearity of RSRP values than the

heuristic approach which uses the path loss model to extract RSRP values from the sheets provided by the antenna manufacturer.

- The prediction error is impacted by the amount of samples taken from the target domain. In particular, the amount of data taken from the target domain can be decreased by up to 90%, if we consider an initial amount of 590 instances taken for training or fine tuning, with a maximum increase in error rate of 2% for the transfer learning approach. It is worth noting that if no data is taken from the target domain, the transfer learning approaches must be used under the assumptions of traditional machine learning, where source and target domain are similar. As this is not the case, DNN T outperforms DNN BS when the amount of instances taken for training is more than 20. This justifies the need to model our problem under the framework of transfer learning in order to decrease the prediction error.

- The transfer learning approach (i.e., DNN T) outperforms the methods that use data from the same tilt configuration (k-NN S, RF S and DNN S) for training, when the amount of samples taken from the testing set is less than 40 instances out of 590. This is because, transfer learning approaches better capture the physical properties of antenna propagation. Thus, being more robust when information is missing from the cross-domain.

- When the number of data samples is larger than 60, transfer learning performs worse than the non-transfer methods. This indicates, more than 60 points chosen uniformly for training a model are enough to capture all the possible patterns (different RSRP values) in a given radio map while achieving good prediction error (see DD values uniformly sampled curve in Figure 4.5 for more than 60 measurements). In contrast, if less than 60 points are taken, which accounts for 12% of the total amount of points initially used for training, there are too few points to capture all the possible patterns. Therefore, using data from $\mathcal{D}_S$ and retraining via transfer learning leads to performance improvement when the number of data samples is less than 12%.

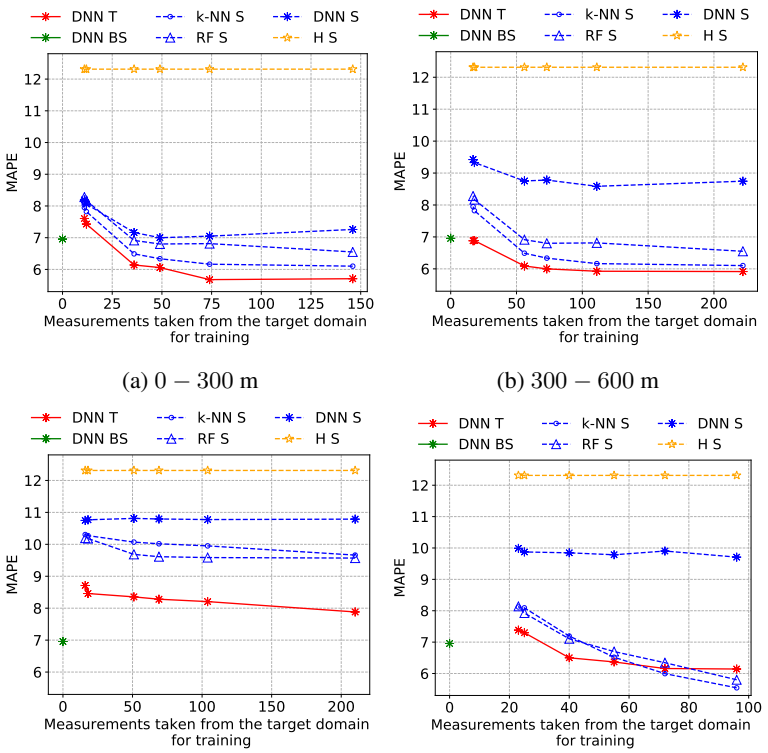
Limited and non-uniformly sampled measurements

We define different antenna distance ranges since we assume to have available only measurements collected in one of those locations. Figure 4.9 shows the obtained average MAPE across all PCIs and all possible combinations of training and testing tilts. Figures 4.9a, 4.9b and 4.9c show the av-

erage MAPE for measurements collected between 0 to 300, 300 to 600, and more than 600 m from the antenna location, respectively. In addition, we consider in Figure 4.9d, the case where we take points from all of the three ranges with probability 0.1, 0.1 and 0.8, respectively. This set of experiments, is motivated by the fact that in a realistic scenario points at a given distance range might be the only ones available to carry out predictions. For instance, in some areas it might not be possible to take measurements due to the existence of obstacles or private properties. In other cases there might be budget constraints (both in terms of resources and time) which do not allow for an extensive measurement campaign of the whole area. These scenarios are particularly challenging, because traditional methods do not work at their best. Therefore we focus on these to highlight the benefits of transfer learning. To study these cases, we consider two possible options: (i) we use the available points as the training set to carry out predictions under the same tilt configuration (i.e., H S, k-NN S, RF S, DNN S) or (ii) we use the available points as part of the retraining step in the transfer learning pipeline (i.e., DNN T). In both cases, the model output is the predicted radio map for the whole area.

We can draw the following conclusions:

- As before, all the machine learning methods (i.e., H S, k-NN S, RF S, DNN S and DNN T) outperform the heuristic approach (i.e., H S) and are impacted by the amount of instances taken from the target domain for training or fine tuning the models.
- The prediction error is never more than 2% higher than when using uniformly sampled data. An increase in error is expected since training and testing sets in the target domain are more dissimilar than when samples are taken uniformly (see DD values in Figure 4.5). However, depending on the application and data restriction when collecting samples, non-uniformly distributed data could still be used to carry out predictions when uniformly sampled data is not available.
- The transfer learning approach (i.e., DNN T) outperforms the methods that use data from the same tilt configuration (i.e., k-NN S, RF S and DNN S) to carry out predictions by a larger margin than with uniformly sampled data. This method is proven to be robust against the bias introduced between training and test sets in the target domain. As such, it performs extremely well when there is a large different between the training and testing sets in the target domain.
- In Figure 4.9d, where points are taken with different probabilities over



(a) $0 - 300$ m

(b) $300 - 600$ m

(c) Above 600 m

(d) Probability 0.1, 0.1 and 0.8

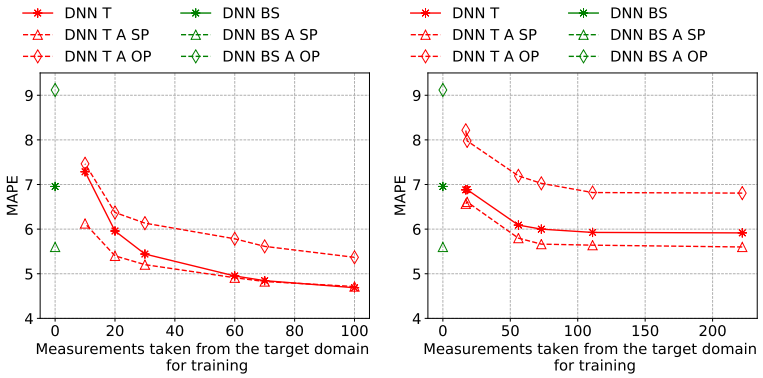
Figure 4.9: MAPE when training or fine tuning on non-uniformly sampled measurements

all distance ranges, the gains of using transfer learning are higher than in the case where the data is uniformly sampled (see Figure 4.8).

In conclusion, the proposed transfer learning approach is able to outperform other methods when using real measurements for both uniformly and non-uniformly sampled data. It has benefits compared to the benchmark methods in both of the following situations: (1) when the available samples of the target domain data are sampled uniformly, but the number of the samples is limited, and (2) when the available measurements used for training or fine tuning on the target domain are non-uniformly sampled. Moreover, considerable accuracy gains are achieved when augmenting the source domain with data coming from other available tilt configurations of the same antenna. This is discussed in the next section.

4.7.2 Tilt Augmentation Transfer

Data augmentation has been shown to be successful in the area of computer vision. By augmenting an existing dataset with new data that follows the same distribution as the data used for training, overfitting can be reduced [53]. In our case, we take inspiration from this idea and we augment the source domain by adding data from other available tilt configurations within the same PCI (i.e., suffix A SP on the graphs below) and from different PCIs (i.e., suffix A OP). We map the obtained MAPE to the DD to analyze the cases where data augmentation improves performance. Table 4.3 shows the DD between the training set in $\mathcal{D}_S$ and the test set in $\mathcal{D}_T$ both before data augmentation and after data augmentation. Data augmentation is performed by either adding data from the same PCI or adding data from the same and different PCIs. Table 4.3 also shows the total amount of training samples used in each case in $\mathcal{D}_S$. It can be noted that, DD values are much higher when using data from different PCIs than in the rest of the cases. This is because adding data from a different PCI will increase the difference between the training set in $\mathcal{D}_S$ and the test set in $\mathcal{D}_T$, therefore overfitting will be increased in $\mathcal{D}_T$. However, the degree of similarity between radio maps coming from the same PCI is higher, thus adding data with a greater similarity to the training set in $\mathcal{D}_S$ and test set in $\mathcal{D}_T$ can help to reduce overfitting and improve accuracy. We evaluate the gains of performing transfer learning from a bigger and more diverse source domain.



(a) Limited uniformly sampled data (b) Limited non-uniformly sampled data

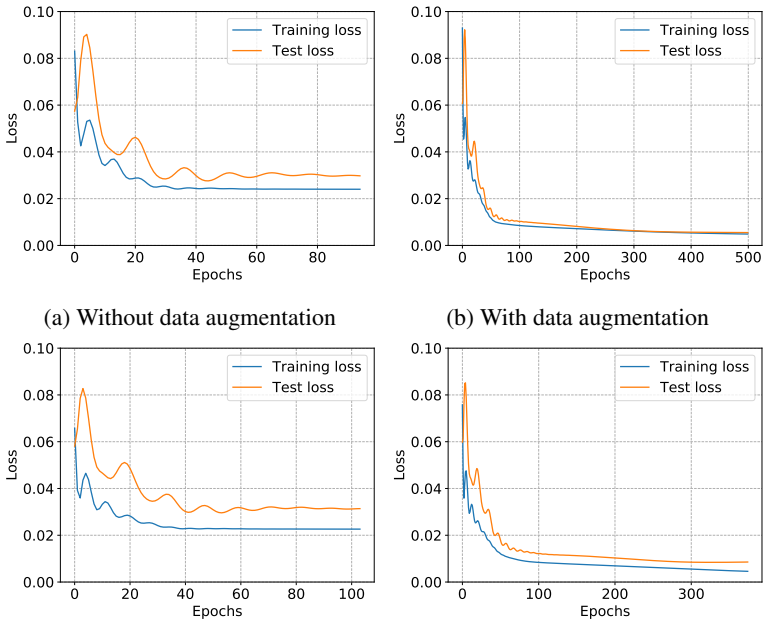
Figure 4.10: MAPE with and without data augmentation

Algorithm	DD	$\mathcal{D}_S$ Number of samples
No augmentation	1.15	600
Augmentation same PCI	1.39	1200
Augmentation other PCIs	3.53	4800

Table 4.3: DD values before and after data augmentation

Figure 4.10 shows the average MAPE across all the PCIs and pairs of training and testing tilt combinations possible when performing data augmentation on the source domain. Figure 4.10a shows the average MAPE when the instances taken from the target domain for training or fine tuning are limited and sampled uniformly. In contrast, Figure 4.10b shows the average MAPE for cases when the measurements taken from the target domain were collected at a distance range from the antenna between 300 and 600 m. We can draw the following conclusions:

- When using data augmentation on the source domain, the prediction error decreases by more than a 1% when the amount of instances taken from the target domain varies between 10 and 40 (see Figure 4.10a).
- In Figure 4.10b we can also observe a performance improvement when compared to the performance achieved without augmenting the source domain.
- In both cases, the performance improvement can be explained by the fact that data augmentation reduces overfitting. Figure 4.11 shows the training and cross validation curves for PCI 1 when Tilts 6 and 2 are used as source and target domains, respectively. Figures 6.3a



(a) Without data augmentation

(b) With data augmentation

(c) Without data augmentation

(d) With data augmentation

Figure 4.11: Training curves on the target domain, PCI 1 and Tilts 6 and 2: (a), (b) Limited uniformly sampled data, (c), (d) Limited non-uniformly sampled data

and 6.3b illustrate the training and cross validation losses without and with data augmentation, respectively, when a limited number of samples are taken uniformly. Figures 6.3c and 6.3f show the training and cross validation losses without and with data augmentation, respectively, when samples are taken at a distance between 300 and 600 m from the antenna location. In both cases, the gap between training and cross validation errors decreases when the source domain is augmented by adding data available from other tilt configurations, from the same PCI. This ensures the transfer learning model is less prone to overfitting.

- Adding data from different tilt configurations from different PCIs (i.e., DNN T A OP) does not lead to performance improvements. This is expected since the DD between the training set in $\mathcal{D}_S$ and the test set in $\mathcal{D}_T$ is much higher (see Table 4.3, DD = 3.53) than in the rest of the cases (see Table 4.3, DD = 1.39), therefore overfitting is more likely to happen.

4.8 Summary

In this chapter, we addressed the problem of predicting the signal strength in the downlink of a real LTE network, where the antennas can be tuned to operate with different antenna tilt configurations. Different approaches were considered as candidates for predicting the signal strength. All of them were based on refined features related to propagation and antenna configuration. As opposed to other works in the field of radio map inference, we studied the quality of prediction of the aforementioned approaches when the datasets used for training and testing are related, but not sampled from the same distribution. We observed that the performance of the predictive models is dependent on the amount of data taken from the testing domain for training or fine tuning. Furthermore, the proposed transfer learning algorithms are shown to be more efficient in cases where the amount of data available from the target tilt configuration is very limited, or available at different distance ranges from the antenna location. Finally, we have shown how augmenting data from the source domain by adding data available from other tilts configurations of the same antenna improves the performance of the proposed transfer learning approaches. Augmenting the source domain decreases the prediction error by 1% when the data available from the target domain for training or fine tuning is limited, or at a distance range between 300 and 600 m from the antenna location.

CHAPTER 5

Applications to Key Performance Indicator Anticipation

In this chapter we focus on the problem of KPI prediction in mobile radio networks.

5.1 Motivation

Anticipating the network performance with high accuracy and low overhead can boost the proactive optimization of mobile radio networks. For instance, channel quality prediction (CQP) has been proposed for efficient resource allocation of video streaming traffic [96]. Similarly, cell load prediction in [56], has been proposed to minimize network element sleeping time.

Traditional approaches to network performance prediction, leverage past information from the network node to predict. This is not always feasible, since the network is constantly evolving by adding new cells for which we have no previous performance history. At the same time, the amount of cells to predict simultaneously is in the order of thousands. It is envisaged that this number will keep growing 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 fourth generation wireless networks (4G) LTE cells operating at different frequencies (i.e. 1.8 GHz and 2.1 GHz) and located in several areas of the city. In particular, we focus our analysis on the channel quality indicator (CQI) and user equipment (UE) KPIs, since they are among the most important KPIs metrics the operator could use to monitor the network status and optimizing accordingly. The CQI reflects the channel status provided by the UEs to their respective cells, ranging from 0 to 15 in LTE. The UE measures the cell load, which is given as the amount of UEs connected to a given cell.

In this chapter, 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-Gig frequency carrier is active at one cell, and the network operator needs to decide whether to activate higher layers 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.

5.2 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 [13]. 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 [105] for CQP. Other studies exploit the nonlinear characteristics of the channel. For instance, in [104] the spectrum sensing process is modeled as a non-stationary Hidden Markov Model. In [57], spatial and temporal correlation are taken into account to model the

CQP as a multivariate Gaussian Process. As for the cell load, similar kinds of studies have been carried out. In [39] the authors use k-Means and SVM classifiers on CDR data to predict the cell load in order to improve network planning. Similarly, in [61], 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 RNN architecture variations that have been shown to be successful for this problem setting. As as example, [110] uses Taguchi optimization, and LSTMs for spectrum prediction, specifically for channel quality as well as channel occupancy. In [9] 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 [62] for 5G, where CNNs and LSTMs are used for making predictions. For a comprehensive overview on channel quality prediction the interested reader may refer to the survey in [13]. The same kind of algorithms have also been used for cell load prediction. For instance, in [6] 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.

5.3 Problem Statement

In this chapter, we propose the application of machine learning algorithms to predict the channel quality and cell load, which is given by the amount 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.

To this end, 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. For each of the 100 cells, 2 time series are recorded. 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 [26] and KPSS [54] 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.

5.3.1 Notation

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 cell c during T hours.

The CQP 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 c, $(x_i)_{i=l}^{t-1}$, we aim to forecast future CQI or UE values $(\hat{y}_i)_{i=t}^{t+N}$, where N denotes the forecasting horizon. For this purpose, we leverage CQI or UE observations from different cells c' chosen according to different selection criteria.

5.4 Prediction Approaches

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

5.4.1 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 reframed 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. Fig. 6.1 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

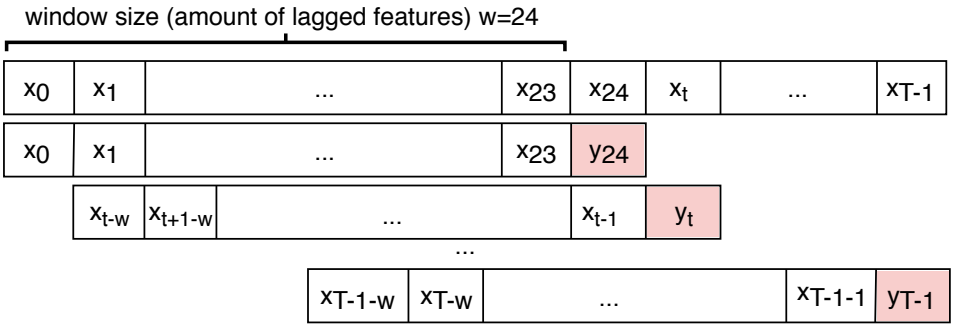


Figure 5.1: Sliding windows for time series forecasting

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 RMSE between real and predicted values in the test set (with size $T_{\text{test}} = N$) defined by Eq. 5.1:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=t}^{t+N}(\hat{y}_i - y_i)^2}. \tag{5.1}$$

Fig. 5.2 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

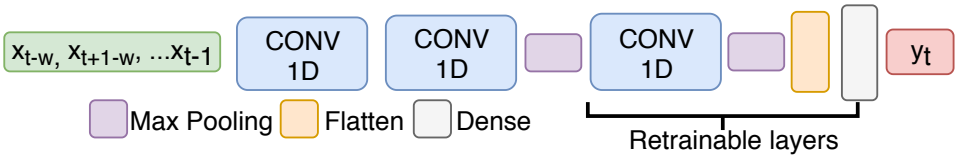


Figure 5.2: CNN architecture

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 casse.

5.4.2 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 Fig. 5.2 a CNN architecture, where we freeze the first 2 convolutional layers and retrain and randomly initialize 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. In the following we describe how D_S cells were selected:

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. Below, 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.

6. Transfer Hybrid Model with Correlated Counters (CNN THM CC): Similar to the CNN TCM but instead 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.

Unsupervised Learning: 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 [49] 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.

5.4.3 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 6.4.3) 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 [11]. 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.

5.5 Experimental Setup

The reference dataset is comprised of 100 time series per KPI. Each time series contains 583 samples with 1 h of time granularity (see Section 5.3). 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, 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.

5.5.1 Parameter Optimization

The different prediction approaches encompass different parameters, which require fine tuning for further optimization. Auto ARIMA requires $m = 24$ to be set *a priori* (Section 6.4.1). 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 5.1: Default combination of hyperparameters

KPI	24	48	96
CQI	**1.70**	1.74	1.74
EU	**1.80**	1.83	1.80

Table 5.2: Lag number input

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 5.1. 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. 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.

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

KPI	100	300	500
CQI	**1.64**	1.70	1.69
EU	1.81	**1.80**	1.83

Table 5.3: Number of epochs

KPI	0.0001	0.001	0.01	0.1
CQI	**1.40**	1.64	1.57	1.75
EU	1.84	**1.80**	2.08	3.31

Table 5.4: Learning rate

KPI	2	3	4
CQI	**1.30**	1.40	1.44
EU	1.94	1.80	**1.71**

Table 5.5: Number of layers

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 5.6.3). Figure 5.3 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.

5.6 Results

5.6.1 City Models Transfer

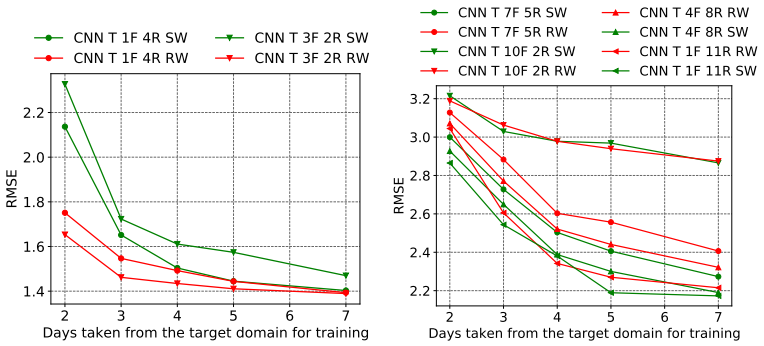
In this section we show the performance of the transfer learning approach for the available 100 cells. Fig. 5.4 shows the average RMSE when applying the different models (see Section 6.4.3 for more details about the different models). The x-axis shows the amount of samples (measured in

KPI	128	256	512
CQI	1.32	**1.30**	1.32
EU	1.71	**1.71**	1.73

Table 5.6: Number of filters

KPI	0	0.2	0.5
CQI	**1.30**	1.30	1.31
EU	1.71	**1.68**	1.89

Table 5.7: Dropout



(a) RSME CQI (b) RSME UE

Figure 5.3: M_T Parameter optimization

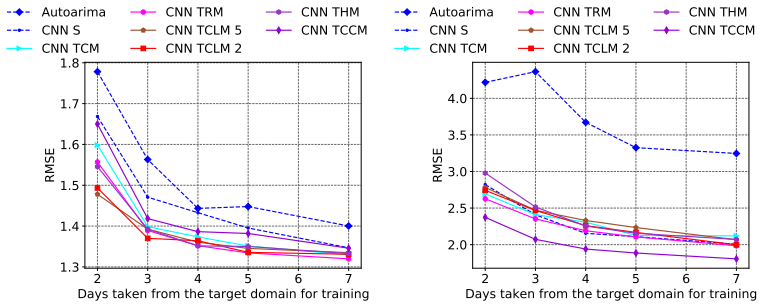
days) available from the target domain that was used for training or fine tuning the models accordingly. The S curves refer to cases where the proposed methods are applied by leveraging only data available from the same cell. The TL curves are related to the transfer learning scenarios, where the model 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.

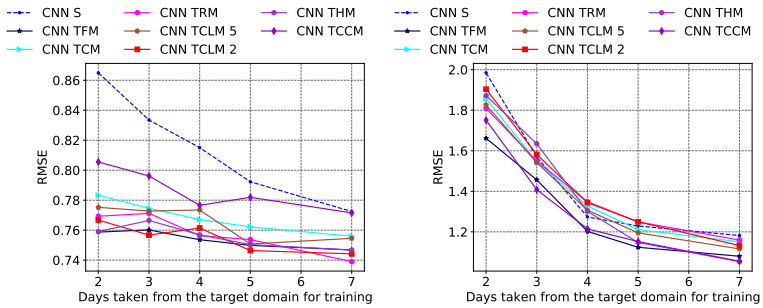
5.6.2 Frequency vs. City Models Transfer

In this section, we compare the city models performance with the CNN TFM performance. The CNN TFM requires that data from the same lo-



(a) RSME CQI (b) RSME UE

Figure 5.4: City models



(a) RSME CQI (b) RSME UE

Figure 5.5: Frequency vs. city models

cation, at a higher frequency, as such, it cannot be used when the higher frequency is turned off. More information is available in Section 6.4.3. Fig. 5.5 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 5.6.3 for more details on model complexity).

5.6.3 Complexity Analysis

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

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 5.8: Trainable parameters

Trainable Parameters

Table 5.8 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 5.8, 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.

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

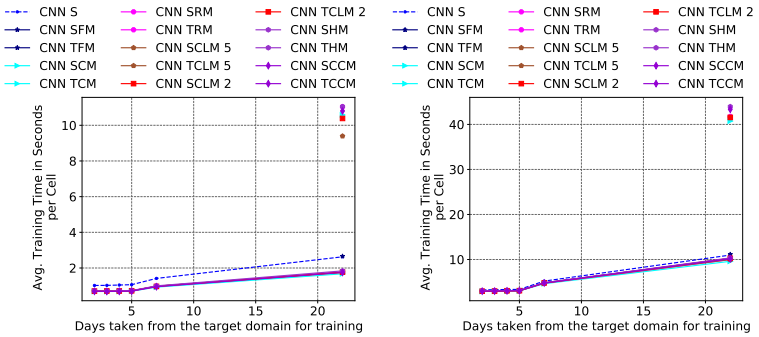
node at the same time. Figures 5.6a and 5.6b show the average training time without transfer learning (i.e., CNN S), the initialization step of transfer learning by training a model on D_S (i.e., CNN SCM, CNN SRM, CNN SCLM, CNN SHM, CNN SRM) and during the retraining step of transfer learning (i.e., CNN TCM, CNN TRM, CNN TCLM, CNN THM, CNN TRM). We can make the following observations:

- Training times for CQI are lower than for UE as the network architecture is smaller.
- In Figure 5.6a 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 5.6b) 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 times, which comprise training time on D_S plus the retraining time on D_T with limited data are 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.

Total Training Time

If predictions are carried out sequentially, training, retraining and inference per cell are performed one after each other. In Table 6.4 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 CNNS, 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.



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

Figure 5.6: Training times per cell in seconds

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

Table 5.9: Sequential training time

- 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.

5.7 Summary

We proposed a transfer learning framework designed to predict CQP 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 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.

CHAPTER 6

Applications to Multi Step Resource Utilization Prediction

In this chapter, we leverage deep and transfer learning algorithms for multi-step resource utilization prediction in mobile radio networks.

6.1 Motivation

Resource utilization predictions are of great importance to anticipate the network status and trigger the optimization pipeline. For instance, accurate Physical Resource Block (PRB) availability prediction has been used to support the 5G network slice broker in making it more capable of provisioning a slice and reducing over-provisioning [38]. Similarly, in [92] the authors propose a Virtual Network Function (VNF) migration algorithm based on the prediction of future resource requirements in 5G networks. In both works, the authors define optimization functions using the predicted values obtained a priori. In contrast, we focus on the anticipation pipeline and the challenges that arise from it.

Traditional approaches to resource utilization prediction leverage the use of different network KPIs such as cell load, network volumes or PRB per-

centage utilization. However, when using KPIs data to monitor the network performance and carrying out predictions at a large scale, two major challenges arise: (i) data efficiency (i.e., data storage and processing) and (ii) time complexity. It is often unfeasible for the operator to record and store data from thousands of cells at every time interval. Moreover, data processing and predictions should be carried out in an efficient manner since some optimization use cases, such as scheduling, require a response time in seconds.

To cope with this complex scenario, we extend the transfer learning approach introduced in Chapter 5, by carrying out resource utilization predictions for longer time intervals in the future and at a large scale in a city.

6.2 Related Work

Wireless network performance, such as traffic load, channel quality or resource utilization, is generally modeled as a time series forecasting problem. There are two predominant approaches:

- **Statistical methods:** Use of traditional statistical methods for time series forecasting. Examples include ARIMA, Seasonal Autoregressive Integrated Moving Average (SARIMA) and nonlinear variations of these methods [89, 115], among others.
- **Machine learning methods**: Use of machine learning, mainly deep learning, by modeling the time series forecasting problem as a sequence prediction problem. For instance, in [106] a Scalable Gaussian Process is used for PRB percentage utilization prediction. Examples of deep learning architectures include RNNs in [80], LSTMs in [38, 101], autoencoders in [99] and CNNs in [109].

6.3 Problem Statement

In this chapter we propose a machine learning approach for the prediction of PRB percentage utilization in radio networks. We leverage past PRB percentage utilization observations for predicting future values at different lookaheads. Each lookahead is defined as the number of time intervals in the future where predictions will be carried out.

6.3.1 Notation

Let $(x_i)_{i=0}^{T-1} = \{x_0, x_1, \ldots, x_{T-1}\}$ be the sequence of PRB utilization values obtained for a given cell $c \in$ during T time intervals, the *multi-step*

PRB prediction problem can be formalized as follows:

Given a limited amount (i.e. $t-1-l$) of PRB utilization observations from a target cell c, $(x_i)_{i=l}^{t-1}$, we aim to forecast future PRB values $(\hat{y}_i)_{i=t}^{t+L-1}$, where L denotes the forecasting horizon referred as lookahead. For the transfer learning task, we leverage PRB utilization observations from a different cell $\hat{c}$, which is chosen according to different criteria based on the Euclidean Distance between c and $\hat{c}$.

6.4 Prediction Approaches

In this section, we describe the prediction approaches used in this work. We also introduce several benchmarks to validate the performance of the proposed transfer learning approach.

6.4.1 Baselines

Most Recent Value

Every PRB sequence of values $(\hat{y}_i)_{i=t}^{t+L-1}$ is predicted with the most recent L available observations $(y_i)_{i=t-L}^{t-1}$.

MLP-E

Each PRB sequence is predicted through a Multi-Layer Perceptron with embeddings. This model takes as inputs: (a) time series data and (b) categorical data. The time series data contains (i) $5 * 8 = 40$ observations corresponding to 5 additional KPIs with window size of 8 and (ii) 3 additional KPIs corresponding to the latest PRB utilization observations from 3 selected neighboring cells. The categorical data includes *day of week*, *quarter hour of day* and *cell index*. These categorical features are mapped to a trainable n-dimensional numerical vectors using the entity embedding technique described in [37]. The output of the model is $5*L$ forecasted KPI values so that one of the KPIs is PRB utilization. The model has 4 layers in total, where each layer contains 64 hidden units. We use ReLU activations and add batch normalization to each layer. In this case, a single model is trained with the data collected from all the cells.

Remark 3. *The MLP-E is a multivariate model with entity embedding trained on all available data from all the cells.*

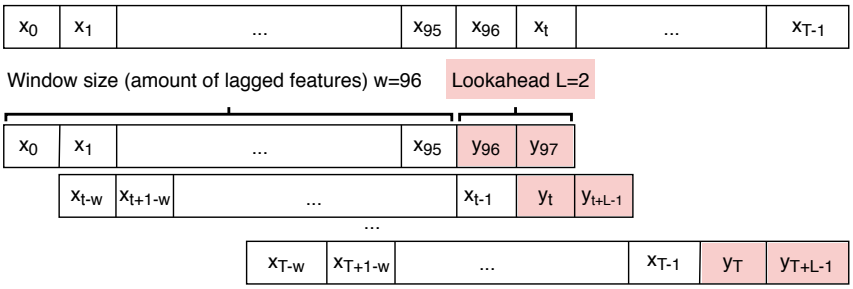


Figure 6.1: Sliding windows for multi step time series forecasting

LSTM-1H

Each PRB sequence is predicted via a multi-variate LSTM model with embeddings. The additional feature embeddings include one-hot encoding of *day of week* and the PRB utilization observations of the 10 highest correlated neighbor cells. Unlike the MLP-E, the LSTM-1H model does not include the cell index as information and needs to be trained per cell.

6.4.2 Multi Step LSTM

First, the general time series forecasting problem is reformulated as a supervised machine learning problem. For this purpose, we use sliding windows of w time intervals by shifting the original time series one step ahead T times. Figure 6.1 shows the process in detail. The resulting supervised machine learning problem is defined as finding the function $f(\mathbf{x}, \boldsymbol{\theta}) = y$, $\mathbf{x} \in \mathbb{R}^w$ and $y \in \mathbb{R}^v$ that maps w time intervals of PRB utilization observations to the PRB values on the next v time intervals, where $v = L$ coincides with the lookeahead. Each input vector $\mathbf{x}$ is given by the sub-sequence $(x_i)_{i=t-w}^{t-1}$, y is given by $(y_i)_{i=t}^{t+L-1}$. As we use neural networks for the task at hand, $\boldsymbol{\theta}$ denotes the neural network weights. The data is divided into training and test sets. Before training, the data is scaled between 0 and 1 by using a min-max scaler. First, the scaler is fitted to the training set and then the resulting function is applied to the test set.

During training, a model $f(\mathbf{x}, \boldsymbol{\theta})$ is created by fitting the selected architecture on the training set. At testing time, $f(\mathbf{x}, \boldsymbol{\theta})$ is applied to data coming from the same or a different cell. Finally, the data is projected to the original space by reversing the scaling.

We choose LSTMs as the building blocks of our architecture, since they shown great performance for sequence prediction. They contain logical gates which make them capable of learning long term dependencies in a sequence. We use a many-to-many LSTM architecture, which is comprised

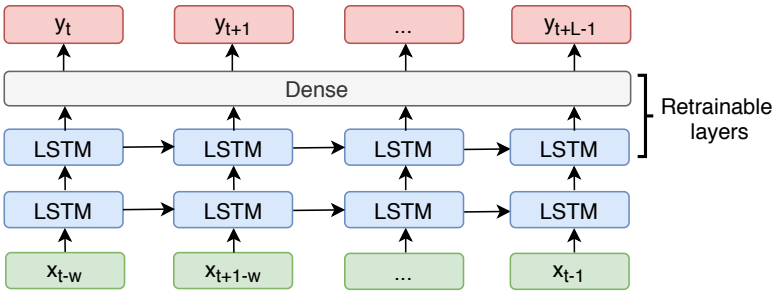


Figure 6.2: LSTM architecture

of 2 LSTM layers followed by 1 dense layer that produces the final output. Figure 6.2 shows the unrolled architecture in time, as well as the inputs and outputs. We use subsequences with a length equal to the selected window size w as inputs, to predict the next v samples in the sequence, which correspond to the LSTM output.

6.4.3 Transfer Learning

Given a source domain D_S with enough data and a target domain D_T with a limited amount of data, to transfer knowledge from D_S to 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 new layers of M_T are going to be either randomly initialized or initialized with M_S weights. Finally, we retrain M_T on D_T. The goals is to transfer the more general features learned by the first layers of the network on a relatively larger source domain (i.e., longer time series) to a limited target domain (i.e., shorter time series). We keep the same architecture for retraining M_T on D_T.

D_S Selection Strategies

For every target cell c in D_T we need to select a source cell $\hat{c}$ on D_S to train M_S. The selection of the source cell depends on the distance measure $ED(x, y)$ between the time series x, y that represent a given pair of cells. The $ED(x, y)$ between these two time series is calculated with Eq. (6.1):

$$\mathrm{ED}(x, y) = \sqrt{\sum_{i=0}^{T-1}(x_i - y_i)^2}. \tag{6.1}$$

We compute $ED(x, y)$ for every pair of cells and select $\hat{c}$ according to one of the following criteria:

- **The closest distance:** $\hat{c}$ is the cell that has the minimum distance from c according to the measure defined in Eq. (6.1) and it is referred to as the closest cell.

- **The most popular cell:** $\hat{c}$ is the "most popular" cell across the whole dataset, which is defined as the one that recurs most often as closest cell with respect to all the other cells.

6.5 Experimental Setup

In this section we describe the set of experiments carried out for the different lookaheads and we analyze the factors that affect the achieved performance. We also comment on the training time and complexity of each algorithm.

The dataset used here is comprised of time series data containing PRB percentage utilization observations from 148 cells with 15-minute granularity for a duration of 13 days in a 4G LTE network. Each time series is divided in training and test sets containing 1152 and 96 samples, corresponding to 12 and 1 days of data, that are used for training and test, respectively.

6.5.1 Performance Evaluation

We use the RMSE between real and predicted values to evaluate the performance of our algorithms (see Eq. (6.2)).

$$\mathrm{RMSE}(\hat{y}_i, y_i) = \sqrt{\frac{1}{N} \sum_{i=t}^{t+N-1} (\hat{y}_i - y_i)^2}. \tag{6.2}$$

6.5.2 Parameters Optimization

To find the "best" possible architecture M_T, we tried several combinations of hyperparameters and selected the one that led to the lowest RMSE across all the cells.

M_S Parameters

Table 6.1 shows the selection of hyperparameters for M_S.

Batch size	128
Number of epochs	50
Total number of layers	2
Activation function	Hyperbolic Tangent (TANH)
Optimizer	Adam
Loss	Mean Squared Error
Learning rate	0.001

Table 6.1: M_S Hyperparameters

M_T Retrainable Layers	Weights Initialization	RMSE
1	From D_S	8.67
1	Random	9.59
2	From D_S	8.51
2	Random	8.77

Table 6.2: M_T Hyperparameters

M_T Parameters Optimization

In M_T, we keep the same architecture of M_S to ensure fair comparisons. The amount of layers to freeze and retrain in M_T tells how much knowledge can be transfer from D_S to D_T. Therefore, we try different combinations of layers to freeze and retrain. Similarly, M_T layers can be either randomly initialized or initialized with pretrained weights in M_S. Table 6.2 shows the RMSE achieved when we used different amount of layers to freeze and retrain on M_T with and without random initialization for $L = 1$ and using the closest cell. It can be observed that the combinations of hyperparameters leading to the best performance is when we retrain the last two layers of M_T and initialize the layers with the pretrained weights from M_S, therefore this is the network setting used for M_T in the rest of the chapter.

6.6 Results

6.6.1 Multi-Step Forecasting

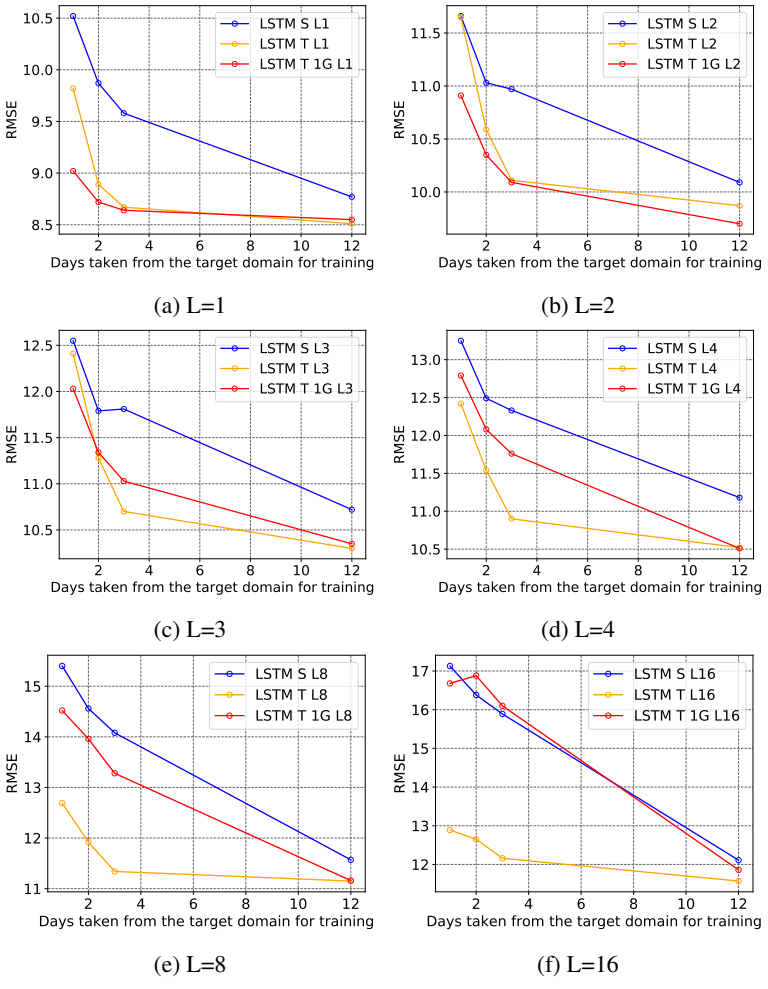
In this section we show the algorithm performance for PRB percentage utilization prediction for different lookaheads. Table 6.3 shows the RMSE for all approaches when using the maximum amount of data available taken from the target domain for training (i.e., 12 days) for all the 148 cells. Figure 6.3 shows the same figures when decreasing this number from 12 to 1 day of training data. In both cases, LSTM S refers to a model trained

L	Most Recent	MLP-E	LSTM-1H	LSTM S	LSTM T	LSTM T 1G
1	9.24	8.30	9.45	8.77	8.51	8.55
2	11.26	8.92	10.33	10.09	9.87	9.7
3	12.42	9.18	10.72	10.72	10.3	10.35
4	13.36	9.36	10.84	11.18	10.52	10.51
8	17.28	9.93	11.43	11.57	11.15	11.53
16	23.43	10.42	11.77	12.11	11.57	11.86

Table 6.3: RMSE per lookahead,
12 days taken from the target domain for training

on data from D_T without using transfer learning. In contrast, LSTM T and LSTM T 1G refer to the transfer learning models when using the closest and the most popular cells to train a model on D_S (see Section 6.4.3), respectively. For all LSTM networks (with or without transfer learning), a window size $w = 96$ (namely, the number of time steps as one of the three input dimensions of LSTM) is used. Each lookahead represents the sequence of values to be predicted in the future. As an example when $L = 1$, we use 1 day of data, corresponding to 96 observations, to predict the value corresponding to the next time interval (i.e., 15 minutes). Similarly, if $L = 3$, we use 1 day of data to predict the sequence of values corresponding to the next 3 time intervals, i.e., 15, 30 and 45 minutes, respectively. From Table 6.3 and Figure 6.3 we can draw the following conclusions:

- All the machine learning methods (i.e. MLP-E and LSTM with or without transfer) outperform the Most Recent Value baseline.
- LSTM S, LSTM T and LSTM T 1G achieved comparable RMSE to the MLP-E and LSTM-1H. This is not surprising for the MLP-E, since it is a single model trained on muti-variate data (i.e., multiple KPIs) from all the cells (cf. Remark 3). In addition, both MLP-E and LSTM-1H have additional features such as the day of the week embedding, a quarter hour embedding and information about neighbor cells. Therefore, the clear benefit of using LSTM S, LSTM T and LSTM 1G over MLP-E lies in the amount of training data required, which can be reduced significantly without sacrificing performance.
- Overall, the RMSE increases for bigger lookaheads. This is reasonable since uncertainty increases over time in the future.
- The transfer learning methods (i.e., LSTM T and LSTM 1G) outperform the non transfer learning method (i.e., LSTM S) for almost all the lookaheads. The difference is more noticeable for LSTM T and



(a) L=1 (b) L=2

(c) L=3 (d) L=4

(e) L=8 (f) L=16

Figure 6.3: RMSE per lookahead

when the amount of data used for training or retraining from D_T is limited.

- Among the transfer learning methods, LSTM T leads to a lower RMSE than LSTM T 1G. This is expected since the ED between D_S and D_T is lower when picking the closest cell as D_S than when using the most popular cell across the whole dataset as D_S (see Section 6.4.3 for more details on D_S selection strategies). However, in terms of complexity LSTM T 1G is more efficient since the amount of models to train on D_S is significantly reduced (see Section 6.6.3 for more details on training time and complexity).

6.6.2 Error Analysis

When carrying out exploratory data analysis on the achieved results, we analyze the factors that affect the transfer learning accuracy (i.e., LSTM T RMSE). In the following, M_T refers to LSTM T and M_S is used to denote the model trained on D_S, which is the first step of the transfer learning pipeline (see Section 6.4.3). Finally, M_L refers to LSTM S and it is used to denote the model trained on the available data from D_T without transfer learning.

Figure 6.4 shows the RMSE when using M_T plotted as a function of both, M_L and the ED to the closest cell. We can draw the following conclusions:

- There is a well defined linear trend between the ED and RMSE when using M_T, which means the closer the distance between the source and the target cell, the smaller the M_T error.
- A similar linear trend can be observed between M_L RMSE and M_T RMSE. This means that if the prediction error in a given time series is already high, we could expect a high transfer learning error even if the similarity between the source and target cell is high.
- In contrast, we could not find any particular trend when plotting the error of training M_S on D_S. Therefore, M_S RMSE does not seem to influence the performance of M_T. Results are excluded here for brevity.
- We can conclude that the ED between D_S and D_T and M_L error are the factors that influence the performance of M_T. Moreover, due to this strong correlation, we fit a linear regression model according to Eq. (6.3) and Eq. (6.4), we find that we can predict M_T RMSE with

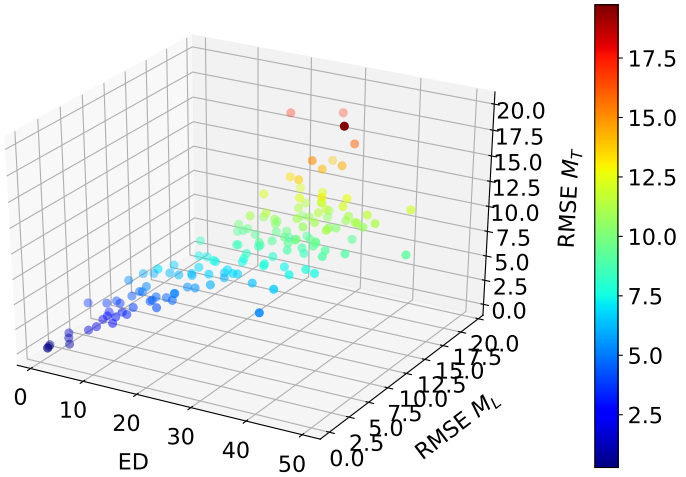


Figure 6.4: Factors affecting LSTM T RMSE

$RMSE = 0.81$ and $R^2 = 0.93$. This is useful to anticipate in $O(1)$ the gains of carrying out the transfer learning retraining step for a new cell in the system.

$$\text{RMSE}(M_T) = f(\text{ED}(D_S, D_T), \text{RMSE}(M_L)), \tag{6.3}$$

$$\text{RMSE}(M_T) = 0.31 - 0.02 * \text{ED}(D_S, D_T) + 0.95 * \text{RMSE}(M_L) \tag{6.4}$$

6.6.3 Complexity Analysis

In this section, we analyze the training time of each approach for the different lookaheads. It is worth observing that, the inference time does not represent a bottleneck in the system. This operation is carried out in $O(1)$ per cell, since it only depends on the sliding window size, which is fixed beforehand. Therefore, the inference time results are not shown here. 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. We use Keras [17] due to its flexibility in implementing the transfer learning approach.

Table 6.4 shows the training duration per cell in seconds, for LSTM S, LSTM T and LSTM T 1G, for the different lookaheads and 12 days taken from D_T for training or retraining.

- It can be observed that the training time is not significantly affected by the lookahead length. This is reasonable since the architecture and

L	LSTM S	LSTM T	**LSTM 1G**	Reduction in Hours
1	$n * 79.85s = 3.28h$	$n * 80.06s + n * 41.47s = 4.99h$	$1 * 80.56s + n * 41.27s = 1.71h$	$1.57h$
2	$n * 80.55s = 3.31h$	$n * 79.76s + n * 41.28s = 4.97h$	$1 * 80.35s + n * 41.21s = 1.71h$	$1.6h$
3	$n * 79.82s = 3.28h$	$n * 79.73s + n * 41.17s = 4.97h$	$1 * 80.83s + n * 41.22s = 1.71h$	$1.57h$
4	$n * 79.76s = 3.27h$	$n * 79.90s + n * 41.40s = 4.98h$	$1 * 80.08s + n * 41.16s = 1.71h$	$1.56h$
8	$n * 79.83s = 3.28h$	$n * 79.65s + n * 41.22s = 4.96$	$1 * 80.50s + n * 41.18s = 1.71h$	$1.57h$
16	$n * 79.08s = 3.25h$	$n * 79.06s + n * 41.18s = 4.94h$	$1 * 79.57 + n * 40.69s = 1.69h$	$1.56h$

Table 6.4: Sequential training time for $n = 148$ cells

amount of training data remain almost the same for the different lookaheads. The most noticeable difference is in the dimension of the last dense layer, which is equal to the lookahead length.

- The time series length, which corresponds to the amount of data used for training LSTM S or retraining LSMT T (see x-Axis in Figure 6.5), considerably impacts the training time. For instance, in Figure 6.5, the training time increases from less than 20 seconds for 2 days to more than 40 seconds for 12 days for both LSTM S and LSTM T.
- The amount of time required for retraining (i.e., LSTM T) is considerably smaller than the amount of time required for training a model without transfer (i.e., LSTM S). This is because, when a model is retrained for transfer learning, the majority of layers are frozen and fewer parameters need to be found (e.g., $49,473$ parameters for LSTM T against $116,033$ parameters for LSTM S per cell, for $L = 1$). This is highly beneficial for architectures where M_S is trained once as part of an initialization phase and retraining needs to be performed often.
- Total training times significantly decrease when we use the most popular cell as D_S (i.e., LSTM T 1G). In this case, we reduce the amount of models to train on D_S and therefore parameters to find for LSTM T with regard to LSTM S. For instance, for $L = 1$ with LSTM S the initial amount of models and parameters required to predict n cells is n and $17,172,884$, respectively. These numbers are nearly identical for LSTM T, since for each cell on D_T, training in D_S and a retraining step on D_T need to be performed every time. In contrast, for LSTM T

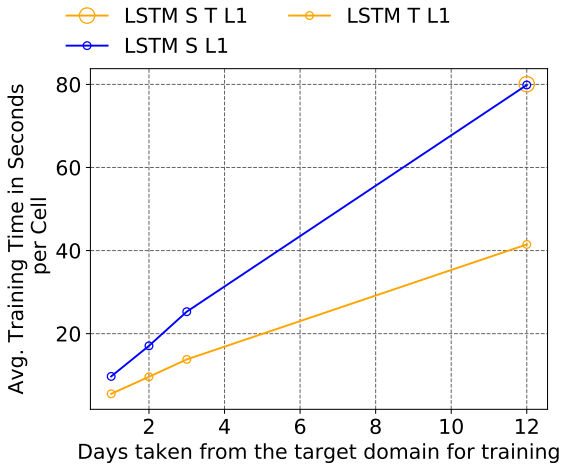


Figure 6.5: Average training time per cell $L = 1$

1G, we only train a model once on D_S and retrain it n times on D_T, thus decreasing the total amount of parameters to $7,438,037$ and the total training time by almost 2 hours.

- Higher gains in terms of training time reduction are expected if the total number of cells to predict is larger than 148. For instance, for 400 cells and lookahead $L = 16$, training time decreases by almost half, from $8.78h$ to $4.53h$.

6.7 Summary

In this chapter, we leveraged the joint use of LSTMs and transfer learning for PRB percentage utilization predictions for the different lookaheads in the future. Our algorithm is shown to achieve state of the art results in terms of accuracy while reducing the required amount of training data. In addition, we nearly halve the amount of training time required by state of the art approaches. Finally, we analysed the factors that led to the achieved accuracy and provide guidelines to select the "best candidate" cells to be used as source domain for the transfer learning task.

Future work will include evaluating the use of embedding and analyzing the relationship between the amount of frozen layers and domain similarity between source and target cells. It is worth noting that, the provided solution could be extended to other network KPIs with minimal parameter optimization.

CHAPTER 7

Conclusions

In this chapter we summarize the main contributions of this thesis and discuss future research directions.

7.1 Conclusions

Accurate, fast and scalable predictions are of vital importance for anticipatory networking tasks. In this thesis, we proposed the joint use of deep and transfer learning for network performance anticipation in mobile wireless networks. Furthermore, we addressed the two major challenges the mobile operator faces when carrying out predictions to anticipate the network performance, namely: (i) Data availability and (ii) Time complexity.

To this end, in Chapter 3, we detail a deep transfer learning framework which we designed to answers the main transfer learning research questions, which are: *What, When and How to transfer?*. We showed how to adapt and optimize several deep learning architectures such as FFNs, CNNs and LSTMs for transfer learning. We also showed the cases where transfer learning leads to improved prediction error, thereby avoiding *negative transfer*. In addition, every time a problem is identified as a possible transfer learning use case we carried out systematic steps such as: EDA,

baselines selection, transfer learning, performance evaluation and finally system deployment (see Section 3.2 for more details on each step).

In all the tested scenarios, we worked with real world data collected from three different commercial 4G LTE networks. In all of them the distribution of the data on source and target domains is different but related, and the source and target tasks are the same. Therefore, our transfer learning algorithms can be seen as an example of transductive learning and parameter transfer, but instead using deep learning (cf. 1 and 2). We tested algorithms in three different real industry problems:

- **Tilt-Dependent Radio Map Prediction:** In Chapter 4, we proposed transfer learning and FFNs to *predict the performance of a given network configuration by leveraging information from different network configurations*. We showed the advantages of the proposed solution when the amount of data available from the target domain is very limited, or non uniformly sampled, which is useful to the operator for reducing drive tests. In addition, we showed how data augmentation can be carried out on the source domain, decreasing the prediction error by 1%. This means there is a trade off between amount of data added to the source domain and domain similarity.

- **Key Performance Indicator Anticipation:** In Chapter 5, we modeled the KPI prediction problem as a univariate time series forecasting problem. The goal is *to predict the value of the CQI or UE for the next hour when a limited amount of past observations are available from the target cell*. For the task at hand, we adapted 1D CNNs and tested several approaches that differ mostly in the way the source domain cells are chosen. Results showed that, for the next hour forecast, there are no significant differences in terms of prediction performance, however using transfer learning significantly decreases the training time, thus there are benefits for the operators in terms of future scalability.

- **Multi Step Resource Utilization Prediction:** In Chapter 6, we focused on a similar problem to Chapter 5, we extended the work in order *to predict the PRB percentage utilization at longer lookaheads*. We developed two different strategies based on a pre-computed ED similarity matrix to select the "best" candidate cells to use as a source domain. This is useful for the operator due to the easy interpretability of the approach, and it being decoupling from the rest of the pipeline as it is a pre-computing step. Results show that, transfer learning outperform the non transfer learning approaches especially at longer

lookaheads. Moreover, the amount of training time required by the transfer learning approach is nearly halved in comparison to the rest of the approaches. In addition, we analyzed the factors that led to the achieved accuracy and proposed an analytical method that can reduce the total training time per cell. Finally, higher gains are expected when the amount of cells where predictions need to be carried out increases, which is a promising result for further scalability.

The main novelty of this work is that, as opposed to other works in mobile wireless networks, where traditional machine learning is applied, we evaluated the performance of the proposed solution when no or a limited amount of data is available from the domain where predictions are carried out. We chose source and target domains based on similarity metrics tailored to the kind of data and Machine Learning (ML) task to be solved. We carry out parameter optimizations for the models in both domains. We then elaborated on and discussed the time complexity of the proposed transfer learning framework with regard to state of the art solutions, specifically when multiple predictions should be carried out in the network simultaneously. Empirical results showed that time complexity can be nearly halved by the use of transfer learning.

Ongoing work is being done in order to test and extend the current solution to other mobile anticipation use cases such as network capacity and cell congestion. The solution is being integrated as part of the Nokia portfolio.

7.2 Discussion and Future work

The application of transfer learning to anticipate mobile network performance is still in early stages. However, throughout this thesis, we have shown that transfer learning is a promising tool to tackle current challenges when carrying out predictions in the network in terms of data availability and time complexity for 5G and beyond. Three main research directions are envisaged for further study. They are summarized below:

- **A different transfer task:** In this thesis, we focused on cases where source and target domains are different but "related" and source and target tasks are the same. The proposed transfer learning framework can be extended and evaluated with minimal modifications to cases where source and target tasks are different. This can be done in two different ways:

  1. *Transfer across KPIs:* DA model could be trained on one KPI (source domain) with sufficient data and retrained using limited

data available from a different KPI (target domain). As an example, a model trained on CQI data can be adapted using a few UE observations to predict UE. It is worth noting that, *negative transfer* could happen if KPIs are too dissimilar. This is useful for the operator in case of having a limited amount of UE observations.

2. *Classification:* The proposed transfer learning approach has been tested for three use cases in mobile wireless networks. For this purpose, different types of data have been used (i.e., spatial data when predicting RSRP values and temporal data for KPI anticipation). As the predicted value has always been continuous, this transfer learning framework has been tested for a ML regression task. However, the same approach could be tailored, with minimal configuration efforts, for classifications tasks. This could be useful for expanding the application of transfer learning to other use cases in mobile wireless networks such as anomaly detection.
3. *Multitask learning:* In Chapter 5, we investigated whether adding different KPIs as new input features to the neural network decreases the prediction error when predicting future values for a single KPI. It was shown that, by using correlated KPIs as additional CNN input features, the RMSE decreases when predicting future UE values. Future work could include the joint use of transfer and multitask learning to predict not only one, but multiple KPIs at the same time. This is a promising approach, since multiple KPIs could be predicted by training one single model, which could decrease time complexity significantly. Based on data availability, other variables (e.g., caching information) could be also used as input features with minimal modifications to the neural network architecture. However, whether this will lead to performance improvement or not requires further investigation in order to avoid *negative transfer*.

- **Enhanced model adaptation:** Well known distance metrics have been used for the selection of source and target domains. The metrics have been tailored to the kind of data and the ML problem to be solved. Further research could include the evaluation of different distance metrics or embedding techniques to study their influence on the prediction error. Similarly, future work could include evaluating the relationship between the amount of knowledge to transfer from the source domain, which is given by the amount of layers to freeze in the target domain model, and the domain similarity between source

and target domains. The main goal of such work would be to find a more systematic and efficient approach to determining the amount of knowledge to transfer. So far this has been accomplished by trying all the possible combinations of layers to freeze and retrain.

- **System deployment and integration:** Finally, the main steps in the anticipation pipeline (i.e., training, retraining and inference) can be deployed as independent services behind an API and orchestrated accordingly. This is useful to give guidelines to the operator on how often new KPI data needs to be collected and stored. Determining the optimal number of cells to train, retrain and predict simultaneously, when carrying out predictions at a large scale, is one of the remaining challenges.

List of Figures

List of Figures

List of Tables

Bibliography

[1] A note on the evaluation of generative models.

[2] 3GPP. Evolved universal terrestrial radio access (e-utra); further advancements for (e-utra) physical layer aspects. *TR 36.814, Techincal report*, 2006.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[4] Evolved Universal Terrestrial Radio Access. Physical layer-measurements (3gpp ts 36.214 version 9.0. 0 release 9). *ETSI TS*, 136(214):V9, 2010.

[5] Farhana Afroz, Ramprasad Subramanian, Roshanak Heidary, Kumbesan Sandrasegaran, and Solaiman Ahmed. Sinr, rsrp, rssi and rsrq measurements in long term evolution networks. *International Journal of Wireless & Mobile Networks*, 2015.

[6] Imad Alawe, Yassine Hadjadj-Aoul, Adlen Ksentinit, Philippe Bertin, César Viho, and Davy 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)*, pages 1–6. IEEE, 2018.

[7] Ejder Baştuğ, Mehdi Bennis, and Mérouane Debbah. A transfer learning approach for cache-enabled wireless networks. In *2015 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 161–166. IEEE, 2015.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[9] Abhijeet Bhorkar, Ke Zhang, and Jin Wang. Deepauto: A hierarchical deep learning framework for real-time prediction in cellular networks. *arXiv preprint arXiv:2001.01553*, 2019.

[10] Oliver Blume, Harald Eckhardt, Siegfried Klein, Edgar Kuehn, and Wieslawa M Wajda. Energy savings in mobile networks based on adaptation to traffic statistics. *Bell Labs Technical Journal*, 15(2):77–94, 2010.

[11] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Bibliography

[13] 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*, 19(3):1790–1821, 2017.

[14] Rita Chattopadhyay, Qian Sun, Wei Fan, Ian Davidson, Sethuraman Panchanathan, and Jieping Ye. Multisource domain adaptation and its application to early detection of fatigue. *ACM Transactions on Knowledge Discovery from Data*, 6(4):18, 2012.

[15] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Artificial neural networks-based machine learning for wireless networks: A tutorial. *IEEE Communications Surveys & Tutorials*, 2019.

[16] Mingzhe Chen, Walid Saad, Changchuan Yin, and Mérouane Debbah. Data correlation-aware resource management in wireless virtual reality (vr): An echo state transfer learning approach. *IEEE Transactions on Communications*, 2019.

[17] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[18] Dan C Cireşan, Ueli Meier, and Jürgen Schmidhuber. Transfer learning for latin and chinese characters with deep neural networks. In *IEEE Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–6, 2012.

[19] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[20] Harshit Daga, Patrick K Nicholson, Ada Gavrilovska, and Diego Lugones. Cartel: A system for collaborative transfer learning at the edge. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 25–37, 2019.

[21] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 193–200, New York, NY, USA, 2007. ACM.

[22] M. Danneberg, J. Holfeld, M. Grieger, M. Amro, and G. Fettweis. Field trial evaluation of ue specific antenna downtilt in an LTE downlink. In *International ITG Workshop on Smart Antennas (WSA)*, pages 274–280. IEEE, 2012.

[23] Hal Daumé III. Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*, 2009.

[24] Jesse Davis and Pedro Domingos. Deep transfer via second-order markov logic. In *ACM 26th annual international conference on machine learning*, pages 217–224, 2009.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

[26] David A Dickey and Wayne A Fuller. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American statistical association*, 74(366a):427–431, 1979.

[27] Lixin Duan, Dong Xu, and Ivor Wai-Hung Tsang. Domain adaptation from multiple sources: A domain-dependent regularization approach. *IEEE Transactions on Neural Networks and Learning Systems*, 23(3):504–518, 2012.

[28] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *ACM SIGKDD*, pages 109–117, 2004.

[29] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Transfer learning for time series classification. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1367–1376. IEEE, 2018.

[30] Wei Feng, Zheng Yan, Hengrun Zhang, Kai Zeng, Yu Xiao, and Y Thomas Hou. A survey on security, privacy, and trust in mobile crowdsourcing. *IEEE Internet of Things Journal*, 5(4):2971–2992, 2017.

[31] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

[32] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[33] Ana Galindo-Serrano, Lorenza Giupponi, and Gunther Auer. Distributed learning in multiuser ofdma femtocell networks. In *IEEE VTC Spring*, pages 1–6, 2011.

[34] Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. Knowledge transfer via multiple model local structure mapping. In *ACM SIGKDD*, pages 283–291, 2008.

[35] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[36] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[37] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*, 2016.

[38] Craig Gutterman, Edward Grinshpun, Sameer Sharma, and Gil Zussman. RAN resource usage prediction for a 5G slice broker. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 231–240. ACM, 2019.

[39] Seif Eddine Hammami, Hossam Afifi, Michel Marot, and Vincent Gauthier. Network planning tool based on network classification and load prediction. In *2016 IEEE Wireless Communications and Networking Conference*, pages 1–6. IEEE, 2016.

[40] Wuri A Hapsari, Anil Umesh, Mikio Iwamura, Malgorzata Tomala, Bódog Gyula, and Benoist Sebire. Minimization of drive tests solution in 3gpp. *IEEE Communications Magazine*, 50(6):28–36, 2012.

[41] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[42] Tin Kam Ho. Random decision forests. In *IEEE Document Analysis and Recognition*, volume 1, pages 278–282, 1995.

[43] Tingting Hou, Gang Feng, Shuang Qin, and Wei Jiang. Proactive content caching by exploiting transfer learning for mobile edge computing. *International Journal of Communication Systems*, 31(11):e3706, 2018.

[44] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 328–339, 2018.

[45] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[46] Chunxiao Jiang, Haijun Zhang, Yong Ren, Zhu Han, Kwang-Cheng Chen, and Lajos Hanzo. Machine learning paradigms for next-generation wireless networks. *IEEE Wireless Communications*, 24(2):98–105, 2017.

[47] Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in nlp. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 264–271, 2007.

Bibliography

[48] James M Joyce. Kullback-leibler divergence. In *International Encyclopedia of Statistical Science*, pages 720–722. Springer, 2011.

[49] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[50] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[51] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[52] Dereje W Kifle, Bernhard Wegmann, Ingo Viering, and Anja Klein. Impact of antenna tilting on propagation shadowing model. In *IEEE VTC Spring*, pages 1–5, 2013.

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[54] Denis Kwiatkowski, Peter CB Phillips, Peter Schmidt, and Yongcheol 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*, 54(1-3):159–178, 1992.

[55] Su-In Lee, Vassil Chatalbashev, David Vickrey, and Daphne Koller. Learning a meta-level prior for feature relevance from multiple related tasks. In *ACM the 24th international conference on Machine learning*, pages 489–496, 2007.

[56] Rongpeng Li, Zhifeng Zhao, Xuan Zhou, and Honggang Zhang. Energy savings scheme in radio access networks via compressive sensing-based traffic load prediction. *Transactions on emerging telecommunications technologies*, 25(4):468–478, 2014.

[57] Qi Liao, Stefan Valentin, and Sławomir Stańczak. Channel gain prediction in wireless networks based on spatial-temporal correlation. In *IEEE SPAWC*, pages 400–404, 2015.

[58] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[59] Xingqin Lin, Jeffrey G Andrews, and Amitabha Ghosh. Modeling, analysis and design for carrier aggregation in heterogeneous cellular networks. *IEEE Transactions on Communications*, 61(9):4002–4015, 2013.

[60] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[61] Enrico Lovisotto, Enrico Vianello, Davide Cazzaro, Michele Polese, Federico Chiariotti, Daniel Zucchetto, Andrea Zanella, and Michele Zorzi. Cell traffic prediction using joint spatio-temporal information. In *2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–4. IEEE, 2017.

[62] Changqing Luo, Jinlong Ji, Qianlong Wang, Xuhui Chen, and Pan Li. Channel state information prediction for 5G wireless communications: A deep learning approach. *IEEE Transactions on Network Science and Engineering*, 2018.

[63] Jun Ma, Jack CP Cheng, Changqing Lin, Yi Tan, and Jingcheng Zhang. Improving air quality prediction accuracy at larger temporal resolutions using deep learning and transfer learning techniques. *Atmospheric Environment*, 214:116885, 2019.

[64] Spyros Makridakis, Steven C Wheelwright, and Rob J Hyndman. *Forecasting methods and applications*. John wiley & sons, 2008.

[65] Lilyana Mihalkova, Tuyen Huynh, and Raymond J Mooney. Mapping and revising markov logic networks for transfer learning. In *AAAI*, volume 7, pages 608–614, 2007.

[66] Jessica Moysen, Lorenza Giupponi, and Josep Mangues-Bafalluy. A mobile network planning tool based on data analytics. *Mobile Information Systems*, 2017, 2017.

[67] Andrew Ng. Nuts and bolts of building ai applications using deep learning.

[68] Huan Cong Nguyen, Ignacio Rodriguez, Troels Bundgaard Sorensen, Jan Elling, Morten Brok Gentsch, Mads Sorensen, and Preben Mogensen. Validation of tilt gain under realistic path loss model and network scenario. In *IEEE VTC Fall*, pages 1–5, 2013.

[69] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *IEEE CVPR*, pages 1717–1724, 2014.

[70] Jialin Pan. *Feature-based transfer learning with real-world applications*. PhD thesis, Hong Kong University of Science and Technology, 2010.

[71] Sinno Jialin Pan, James T Kwok, Qiang Yang, and Jeffrey Junfeng Pan. Adaptive localization in a dynamic WiFi environment through multi-view learning. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 1108–1113. AAAI, 2007.

[72] Sinno Jialin Pan, Dou Shen, Qiang Yang, and James T Kwok. Transferring localization models across space. In *AAAI*, pages 1383–1388, 2008.

[73] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2011.

[74] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[75] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[76] Claudia Parera, Alessandro EC Redondi, Matteo Cesana, Qi Liao, Lutz Ewe, and Cristian Tatino. Transferring knowledge for tilt-dependent radio map prediction. In *IEEE WCNC*, pages 1–6, 2018.

[77] Qihang Peng, Andrew Gilman, Nuno Vasconcelos, Pamela C Cosman, and Laurence B Milstein. Robust deep sensing through transfer learning in cognitive radio. *IEEE Wireless Communications Letters*, 9(1):38–41, 2019.

[78] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[79] Caleb Phillips, Douglas Sicker, and Dirk Grunwald. A survey of wireless path loss prediction and coverage mapping methods. *IEEE Communications Surveys & Tutorials*, 15(1):255–270, 2013.

[80] Chen Qiu, Yanyan Zhang, Zhiyong Feng, Ping Zhang, and Shuguang Cui. Spatio-temporal wireless traffic prediction with recurrent neural network. *IEEE Wireless Communications Letters*, 7(4):554–557, 2018.

[81] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[82] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. In *Advances in Neural Information Processing Systems*, pages 3342–3352, 2019.

Bibliography

[83] Mauro Ribeiro, Katarina Grolinger, Hany F ElYamany, Wilson A Higashino, and Miriam AM Capretz. Transfer learning with seasonal and trend adjustment for cross-building energy forecasting. *Energy and Buildings*, 165:352–363, 2018.

[84] Ignacio Rodriguez, Huan C Nguyen, Troels B Sørensen, Jan Elling, Morten B Gentsch, Mads Sørensen, Lauri Kuru, and Preben Mogensen. A geometrical-based vertical gain correction for signal strength prediction of downtilted base station antennas in urban areas. In *IEEE VTC Fall*, pages 1–5, 2012.

[85] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, 2019.

[86] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[87] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[88] Yifei Shen, Yuanming Shi, Jun Zhang, and Khaled B Letaief. Transfer learning for mixed-integer resource allocation problems in wireless networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[89] Yantai Shu, Minfang Yu, Oliver Yang, Jiakun Liu, and Huifang Feng. Wireless traffic modeling and prediction using seasonal arima models. *IEICE transactions on communications*, 88(10):3992–3999, 2005.

[90] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[91] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[92] Lun Tang, Xiaoyu He, Peipei Zhao, Guofan Zhao, Yu Zhou, and Qianbin Chen. Virtual network function migration based on dynamic resource requirements prediction. *IEEE Access*, 7:112348–112362, 2019.

[93] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[94] Tatiana Tommasi, Francesco Orabona, and Barbara Caputo. Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In *IEEE CVPR*, pages 3081–3088, 2010.

[95] Tatiana Tommasi, Francesco Orabona, and Barbara Caputo. Learning categories from few examples with multi model knowledge transfer. *IEEE transactions on pattern analysis and machine intelligence*, 36(5):928–941, 2014.

[96] Dimitrios Tsilimantos, Amaya Nogales-Gómez, and Stefan Valentin. Anticipatory radio resource management for mobile video streaming with linear programming. In *IEEE ICC*, pages 1–6, 2016.

[97] Stijn Van Dongen and Anton J Enright. Metric distances derived from cosine similarity and pearson and spearman correlations. *arXiv preprint arXiv:1208.3145*, 2012.

[98] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[99] Jing Wang, Jian Tang, Zhiyuan Xu, Yanzhi Wang, Guoliang Xue, Xing Zhang, and Dejun Yang. Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[100] Wei Wang, Jin Zhang, and Qian Zhang. Transfer learning based diagnosis for configuration troubleshooting in self-organizing femtocell networks. In *IEEE GLOBECOM*, pages 1–5, 2011.

[101] Xu Wang, Zimu Zhou, Fu Xiao, Kai Xing, Zheng Yang, Yunhao Liu, and Chunyi Peng. Spatio-temporal analysis and prediction of cellular traffic in metropolis. *IEEE Transactions on Mobile Computing*, 2018.

[102] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016.

[103] Christopher Xie, Alex Talk, and Emily Fox. A unified framework for missing data and cold start prediction for time series data. In *Advances in neural information processing systems Time Series Workshop*, 2016.

[104] Xiaoshuang Xing, Tao Jing, Yan Huo, Hongjuan Li, and Xiuzhen Cheng. Channel quality prediction based on bayesian inference in cognitive radio networks. In *IEEE INFOCOM*, pages 1465–1473, 2013.

[105] Xiang Xu, Mingjian Ni, and Rudolf Mathar. Improving QoS by predictive channel quality feedback for LTE. In *IEEE SoftCOM*, pages 1–5, 2013.

[106] Yue Xu, Feng Yin, Wenjun Xu, Jiaru Lin, and Shuguang Cui. Wireless traffic prediction with scalable Gaussian process: Framework, algorithms, and verification. *IEEE Journal on Selected Areas in Communications*, 37(6):1291–1306, 2019.

[107] Rui Ye and Qun Dai. A novel transfer learning framework for time series forecasting. *Knowledge-Based Systems*, 156:74–99, 2018.

[108] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[109] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional neural network: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[110] Ling Yu, Jin Chen, Guoru Ding, Ya Tu, Jian Yang, and Jiachen Sun. Spectrum prediction based on Taguchi method in deep learning with long short-term memory. *IEEE Access*, 6:45923–45933, 2018.

[111] Alessio Zappone, Marco Di Renzo, and Mérouane Debbah. Wireless networks design in the era of deep learning: Model-based, ai-based, or both? *arXiv preprint arXiv:1902.02647*, 2019.

[112] Chuanting Zhang, Haixia Zhang, Jingping Qiao, Dongfeng Yuan, and Minggao Zhang. Deep transfer learning for intelligent cellular traffic prediction based on cross-domain big data. *IEEE Journal on Selected Areas in Communications*, 37(6):1389–1401, 2019.

[113] Yige Zhang, Aaron Yi Ding, Jörg Ott, Mingxuan Yuan, Jia Zeng, Kun Zhang, and Weixiong Rao. Transfer learning-based outdoor position recovery with telco data. *IEEE Transactions on Mobile Computing*, 2020.

[114] Vincent Wenchen Zheng, Evan Wei Xiang, Qiang Yang, and Dou Shen. Transferring localization models over time. In *AAAI*, pages 1421–1426, 2008.

[115] Bo Zhou, Dan He, Zhili Sun, and Wee Hock Ng. Network traffic modeling and prediction with arima/garch. In *Proc. of HET-NETs Conference*, pages 1–10, 2005.