On Heterogeneous Transfer Learning for Improved Network Service Performance Prediction

Fernando García Sanz*, Masoumeh Ebrahimi*†, and Andreas Johnsson*‡
* Ericsson Research, Sweden, Email: {*fernando.a.garcia.sanz, masoumeh.ebrahimi, andreas.a.johnsson*}*@ericsson.com*
† Uppsala University, Department of Information Technology, Sweden, Email: *andreas.johnsson@it.uu.se*
‡ KTH, Division of Electronics and Embedded Systems, Sweden, Email: *mebr@kth.se*

***Abstract*—Transfer learning has been proposed as an approach for leveraging already learned knowledge in a new environment, especially when the amount of training data is limited. However, due to the dynamic nature of future networks and cloud infrastructures, a new environment may differ from the one the model is trained and transferred from.**

In this paper, we propose and evaluate an approach based on neural networks for heterogeneous transfer learning that addresses model transfer between environments with different input feature sets, which is a natural consequence of network and cloud re-orchestration. We quantify the transfer gain, and empirically show positive gain in a majority of cases. Further, we study the impact of neural-network architectures on the transfer gain, providing tradeoff insights for multiple cases. The evaluation of the approach is performed using data traces collected from a testbed that runs a Video-on-Demand service and a Key-Value Store under various load conditions.

***Index Terms*—Service Performance, Machine Learning, Heterogeneous Transfer Learning.**

I. Introduction

A promising approach enabling intelligent network and service management is the use of machine-learning models that can predict and forecast the service performance based on available measurements and other observations in the network and cloud infrastructure. The ability to learn performance models simplifies management and operational tasks such as service on-boarding, network resource adaptation, proactive service assurance, and root-cause analysis.

Traditionally, many machine-learning applications operate under the assumption that train and test data are both under the same feature space, and that they are extracted from the same distribution [1]. Hence, when the data distribution or its dimensionality changes, the previously crafted models must be rebuilt from scratch, and trained with the new data. Nevertheless, collecting additional data for training new models is expensive and sometimes infeasible. This gets more accentuated in environments with real-time monitoring requirements, or limited computational or networking resources. An example of an environmental change is illustrated in Figure 1 where clients are accessing a cloud service over a network, that is being migrated and scaled from the source to the target domain. This may change the availability of features and thus the feature space to be leveraged by the ML model.

Transfer learning is a mitigation approach where knowledge learned in one environment can be leveraged in another. This enhances the training process, as it is not necessary to develop new models from scratch, rather using previously built models and adapt them to a new environment.

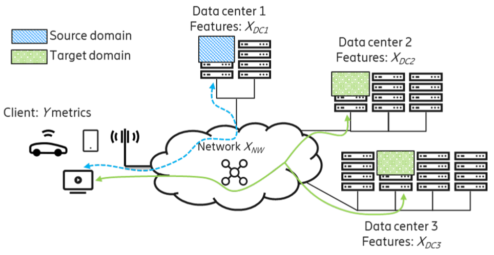


Fig. 1. Prediction of service metrics Y at the client given observations of network and cloud infrastructure state and utilization $\{X_{NW}, X_{DC1..3}\}$. The execution environment changes from source to target domain.

In this paper, which builds upon our previous work on transfer learning for service performance prediction [2], [3], we study the feasibility of transfer learning in a heterogeneous environment where source and target domains do not share a common input feature space. Such changes may occur due to changes in monitoring configurations, feature availability, or added number of features stemming from the horizontal scaling of the service in the target domain as exemplified by Figure 1. More specifically, we study the impact of adding, removing, and changing input features in the target domain on the transfer gain.

The main contributions of this paper are: (1) an approach for heterogeneous transfer learning for service performance modeling, (2) an analysis indicating feasibility and robustness of the approach, and (3) insights on the tradeoff concerning shallow, deep-wide, and deep-narrow neural networks.

II. Transfer Learning Formulation

Figure 1 illustrates the scenario that is considered in this work. Clients are interacting with services that are executing in the cloud. In this paper, we consider data traces originating from testbed experiments where clients access two network services executing in one data center; a Video-on-Demand (VoD) service and a Key-Value Store (KVS) service.

The learning task is to predict the service-level metrics Y, e.g., response time, at time t on the clients accessing the services based on knowing the infrastructure metrics X, e.g., CPU utilization, at time t. We train and evaluate models $M : X_t \rightarrow \hat{Y}_t$, such that $\hat{Y}_t$ closely approximate Y_t for a given

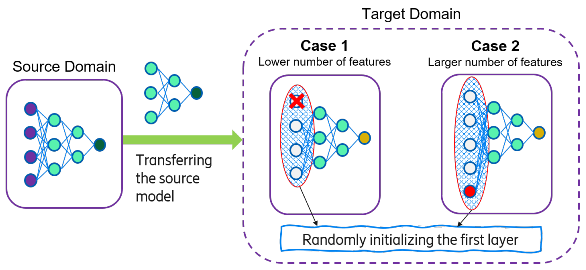

Target Domain
Source Domain
Case 1
Lower number of features
Case 2
Larger number of features
Transferring the source model
Randomly initializing the first layer

Fig. 2. Two cases of a heterogeneous transfer learning approach.

X_t using supervised machine learning. In our previous work [2], [3], we proposed and evaluated approaches using transfer learning to mitigate model performance reduction stemming from a selection of execution environment changes.

Based on the definition of transfer learning in [1], a domain $D = \{X, P(X)\}$ consists of two components: (1) a feature space X, and (2) a marginal probability distribution $P(X)$, where X corresponds to the infrastructure metrics. The number of features in X is denoted $|X|$. Further, a task $T = \{Y, M\}$ consists of two components: (1) a target space Y corresponding to service-level metrics, and (2) an objective predictive model M. Transfer learning is then defined as follows. Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims at reducing the cost of learning the predictive model M in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$. A model that is transferred from D_S to D_T is denoted $M_{S\to T}$, to separate it from a model M_S or M_T that is trained in isolation in the source or target domain.

III. Heterogeneous Transfer Learning

In this section, we explain our transfer learning approach with regard to the heterogeneity in the input feature space. In addition to the heterogeneity aspect, we also elaborate on the impact of feed-forward neural network architecture choices. We vary the configuration in terms of the number of layers and neurons per layer. Further in the result section, we investigate the impact of a varying number of available samples N_t (N at time t) in D_T, as the number of samples in the target domain may be limited due to overhead or time constraints.

A. *Heterogeneous feature space*

Heterogeneous transfer learning (HTL), in this paper, corresponds to a scenario where the available input features X are not identical when the execution environment changes. This may occur due to, for example, changes in monitoring configurations, or other re-orchestration actions.

This paper studies, quantifies, and explains the impact of transferring a source domain to a target domain where $X_S \neq X_T$. We define two heterogeneity cases: (1) the number of features in the source is larger than or equal to the target domain $|X_S| \geq |X_T|$, and (2) the number of features in the target is larger than or equal to the source domain $|X_S| \leq |X_T|$. The two cases are illustrated in Figure 2. In the result section, we examine how much heterogeneity is acceptable while still getting a positive transfer gain.

B. *Model transfer strategy*

The source and target models can be generated separately using the available samples in their domains. For HTL, we suggest transferring the model from the source domain with partial weight transfer. These three neural-network models are described as follows:

Source model (M_S): The model consists of an input layer, corresponding to features X, $n-1$ hidden layers $L_1, ..., L_{n-1}$, weights $w_1, ..., w_n$, and an output layer L_n corresponding to Y_t. The weights w_i for a model M_S are trained using backpropagation [4] with samples in the source domain D_S.

Scratch target model (M_T): The same neural-network architecture is used in the target domain but all weights are randomly initialized, and trained using the available samples in the target domain.

Target model with transferred weights ($M_{S\to T}$): This model builds upon re-training of a feed-forward neural-network model M, that is transferred from a source domain [5]. For heterogeneous transfer learning, the knowledge transfer corresponds to training a target model where the weights of layers $L_2, ..., L_n$ are initialized with the weights from the source model M_S. Further, layer L_1, which now has a different number of weights, is randomly initialized to encompass the feature heterogeneity, as illustrated in Figure 2. After knowledge transfer, the model is denoted $M_{S\to T}$. The target model is then fine-tuned using available samples from the target domain D_T.

C. *Neural network architectures*

Heterogeneous transfer learning directly affects the first layer of the transferred model. This is due to the fact that the number of input features in the source domain is different from the target domain. To overcome this heterogeneity, we randomly initialize the weights of the first neural network layer. This is required to match the target input space that may be larger or smaller than the source domain input space.

We investigate the impact of the neural network architecture on transfer gain in heterogeneous transfer learning. For this purpose, we examine three neural network settings, called shallow, deep-wide, and deep-narrow, as shown in Figure 3, where 4 and 10 stand for the number of layers.

The shallow neural network is composed of a few layers. Although this model may work well in the source domain, it may not be the case in the HTL scenario when the learned model is transferred to the target domain (to train model $M_{S\to T}$). This is due to the fact that the first layer constitutes a large proportion of the total number of parameters. So, a significant amount of knowledge will be lost when re-initializing the weights of the first layer. We argue that a deeper network would trap the knowledge in the middle and last layers of the neural network, which could be successfully transferred to the target domain. Thereby, we propose using

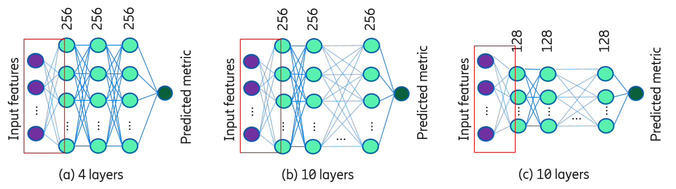


Fig. 3. Neural-network model architectures: (a) shallow-4, (b) deep-wide-10, (c) deep-narrow-10.

a deeper network when transferring models in heterogeneous transfer learning. The hypothesis is that this should provide an enhanced information transfer across domains, improving the performance of the predictive model in the target one. However, simply increasing the network depths (i.e., deep-wide neural network) comes at the cost of an increased number of trainable parameters, and thus the training time. Thereby, we further try to optimize the deep network, and for this, we keep the number of parameters similar to the shallow network, and just spread neurons over more layers, reaching a deep-narrow neural network. Different neural network choices and their number of parameters, examined in our experiments, are listed in Table II.

D. Quantifying the transfer gain

In addition to studying the performance of M_S, M_T, and $M_{S\to T}$, we use the concept of *transfer gain* $\mathcal{G}$ to quantify the impact of D_T on $M_{S\to T}$, which we define as:

$$\mathcal{G} = e_T - e_{S\to T}, \quad (1)$$

where e_T and $e_{S\to T}$ are the model errors for M_T and $M_{S\to T}$, respectively. The transfer gain is inspired by the negative transfer gap defined in [6]. Transfer gain occurs when the loss of the transferred model $M_{S\to T}$ is higher than M_T, whereas a negative transfer gap means the opposite.

IV. Testbed and Data Traces

The evaluation in this paper is based on realistic traces[1] obtained from a testbed. A brief overview of the scenarios and experimental infrastructure are provided below, and additional details are available in our previous work [7].

The testbed consists of a server cluster and a set of clients, all deployed on a rack with ten high-performance machines interconnected by a Gigabit Ethernet. On the server cluster we run two network services: Video-on-Demand (VoD) and Key-Value Store (KVS). The *VoD service* uses a modified VLC media player software, which provides single-representation streaming with a varying frame rate. Further, the *KVS service* uses the Voldemort software. The two services are installed on the same machines and can execute in parallel. The client machines act as load generators for both services.

[1]https://github.com/foroughsh/KTH-traces

TABLE I
Examples of traces obtained from testbed.

Trace ID	Service(s)	Load pattern	Target Y	# samples
K1P	KVS	Periodic	*RAvg, WAvg*	28962
K1F	KVS	Flashcrowd	*RAvg, WAvg*	19444
K2P	KVS + VoD	Periodic	*RAvg, WAvg*	26488
K2F	KVS + VoD	Flashcrowd	*RAvg, WAvg*	24225

A. Generating load on the testbed

Two load generators are running in parallel, one for the VoD application and another for the KVS application, aiming at emulating real traffic scenarios. The VoD load generator controls the number of active VoD sessions, spawning and terminating VLC clients. The KVS load generator controls the rate of KVS operations issued per second.

Both generators produce load according to two distinct load patterns: (1) *periodic-load* where the load generator produces requests following a Poisson process whose arrival rate is modulated by a sinusoidal function, and (2) *flashcrowd-load* where the requests are following a Poisson process whose arrival rate is modulated by a flashcrowd model [8].

B. Collected data and traces

Data traces from the testbed contain an input feature set X and the specific service-level metrics Y_{VoD} and Y_{KVS}. A trace is generated by extracting, and collecting, these statistics during execution of experiments with different configurations.

The feature set X is extracted from the Linux kernels that run on the machines. To access the kernel data, we use System Activity Report (SAR), a popular open-source Linux library, which provides approximately 1700 features per server. Examples of such statistics are CPU utilization per core, memory utilization, and disk I/O.

For the purpose of this paper, we focus on modeling the KVS service where the service-level metrics Y_{KVS} are measured on the clients. During an experiment two main metrics are captured, namely the Read Response Time as the average read latency for obtaining responses over a set of operations performed per second (*RAvg*), and a corresponding metric for the Write Response Time (*WAvg*). The metrics are computed using a customized benchmark tool of Voldemort.

A summary of selected traces captured in the testbed is available in Table I. The trace ID is encoded according to service under investigation (KVS or VoD), number of concurrent services (1 or 2), and load pattern (periodic or flashcrowd).

C. Creation of heterogeneous traces

To study heterogeneous transfer learning for the scenarios considered in this paper, the set of features in the source and target domains must be different. In a first step, we reduce the feature space to 18 features, using domain knowledge, following the approach in [9]. Then, we artificially reduce the number of available features utilizing the Pearson correlation (see Figure 4) between the features and *RAvg* and *WAvg*. For more details about the features see [7].

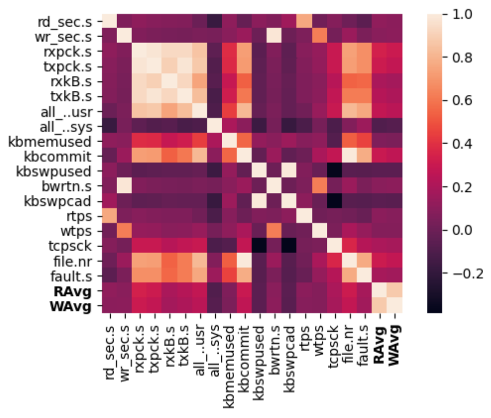


Fig. 4. Feature correlation with *RAvg* and *WAvg*.

TABLE II
NUMBER OF PARAMETERS IN EACH OF THE NETWORKS EMPLOYING 18 FEATURES IN THE SOURCE DOMAIN AND 18 IN THE TARGET DOMAIN.

Network	L_1 parameters	$L_2, ..., L_n$ parameters	Total parameters	Training time (sec)
shallow-4	4864	131841	136705	1.48
deep-wide-6	4864	263425	268289	1.62
deep-wide-10	4864	526593	531457	1.79
deep-narrow-6	3458	133407	136865	1.45
deep-narrow-10	2432	132225	134657	1.53

For the case with lower number of features in the target, features have been iteratively removed, starting with features with a higher correlation with the given task (*RAvg* or *WAvg*). In the case of having more features in the target domain, features in the target are selected from the least to the most correlated, with the intention of creating challenging scenarios. For instance, the most correlated feature with both *RAvg* and *WAvg* is *rxpck.s*, and the second most correlated ones are *file.nr* and *txpck.s*, respectively.

V. Results and Discussion

We evaluate the heterogeneous transfer learning approach, explained in Section III, on a set of transfer scenarios described in Table III, and quantify the impact of the feature space heterogeneity and architectural neural-network design choices.

A source domain D_S, and its corresponding model M_S, are created for each transfer scenario. Further, a target domain corresponds to a change in feature space, i.e., $X_S \neq X_T$ and possibly a change in the prediction task, i.e., $T_S \neq T_T$. This refers to a scenario where the service provider would like to predict either read or write response time for a KVS service in a domain where the set of features has changed.

The *Normalized Mean Absolute Error (NMAE)* is used for quantifying model performance, and is defined as:

$$e = \frac{1}{\bar{y}}(\frac{1}{m}\sum_{t=1}^{m}|y_t - \hat{y}_t|), \quad (2)$$

TABLE III
THE HTL SCENARIOS STUDIED IN THIS WORK. ALL SCENARIOS ARE BASED ON TRACE K2P IN TABLE I. FURTHER, $P(X_S) = P(X_T)$.

#	D_S		D_T		$M_S = M_T$
	$\|X_S\|$	T_S	$\|X_T\|$	T_T	
1	18	*RAvg*	18, 10, 2	*RAvg*	shallow-4
2	2	*RAvg*	18, 10, 2	*RAvg*	shallow-4
3	18	*RAvg*	18, 10, 2	*WAvg*	shallow-4
4	2	*RAvg*	18, 10, 2	*WAvg*	shallow-4
5	18	*RAvg*	2	*WAvg*	shallow-4, deep-wide-6/10
6	2	*RAvg*	18	*WAvg*	shallow-4, deep-wide-6/10
7	18	*RAvg*	2	*WAvg*	shallow-4, deep-narrow-6/10
8	2	*RAvg*	18	*WAvg*	shallow-4, deep-narrow-6/10

where $\hat{y}_t$ is the model prediction for the measured performance metric y_t, and $\bar{y}$ is the average of the samples y_t of the test set of size m.

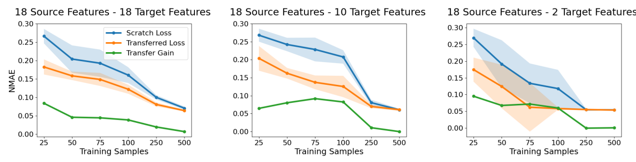
The neural network architectures considered in this work are listed in Table II and partly shown in Figure 3. Further, the implementation of the neural networks has been performed by means of the Keras library[2] running on top of TensorFlow[3]. For the source models, the neural networks are initialized with random weights and are trained on samples (X, Y). We use the ReLU activation function for all the layers. The other configurations are as: Adam optimizer [10] with a learning rate starting at 0.001 using exponential decay with the decay rate of 85, 1000 decay steps and staircase function, and mean absolute error (MAE) as the loss function. Each experiment was run for a maximum of 200 epochs, with early stopping using patience of 10 epochs and weight restoring. Regarding the data, a batch size of 32 was used, performing a validation split of the 20% of the training set, composed by the 80% of the trace samples, being the test set constituted by the remaining 20%.

Similarly, in the target domain, the samples are also split. The training set size is varied between 25 and 500 samples, and it is used to train the transferred model $M_{S\rightarrow T}$ and the scratch model M_T to enable calculation of transfer gain. Both models are trained and tested on the same samples.

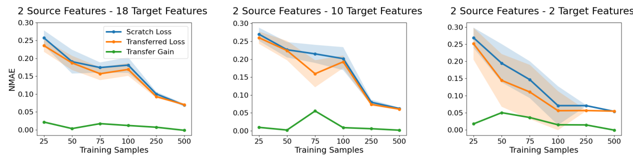
A. Feature space heterogeneity

In Scenarios 1 and 2, defined in Table III, we investigate the performance of the approach for the case when the prediction task is *RAvg* (i.e., $T_S = T_T$), the number of features is different in the source and target domains (i.e., $|X_S| \neq |X_T|$), and the model is of type shallow-4. In addition, all experiments are evaluated using the *K2P* trace (see Table I). In Figure 5a, 18 input features are used in the source domain while we reduce the number of features to 10 and then 2 in the target domain. In Figure 5b, we keep the number of features fixed at 18 in the target and vary the number of features in the source domain.

[2]https://keras.io/
[3]https://github.com/tensorflow/tensorflow

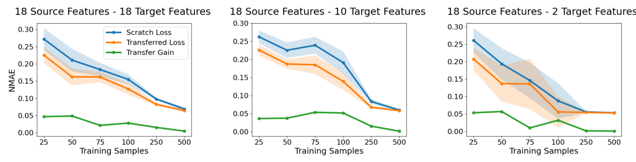


(a) Scenario 1: D_S (18 features, RAvg) and D_T (varying features, RAvg).

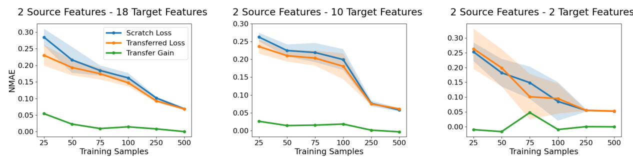


(b) Scenario 2: D_S (2 features, RAvg) and D_T (varying features, RAvg).

Fig. 5. NMAE measurement for different sets of features and the same prediction task in the shallow-4 network.

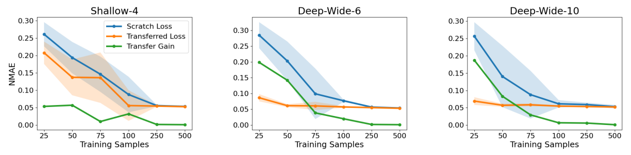


(a) Scenario 3: D_S (18 features, RAvg) and D_T (varying features, WAvg).

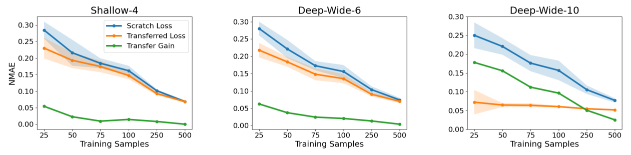


(b) Scenario 4: D_S (2 features, RAvg) and D_T (varying features, WAvg).

Fig. 6. NMAE measurement for different sets of features and different prediction tasks in a shallow-4 network.

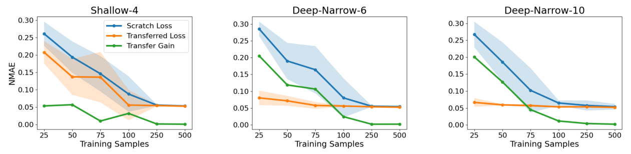


(a) Scenario 5: D_S (18 features, RAvg) and D_T (2 features, WAvg).

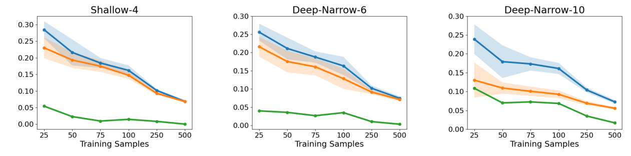


(b) Scenario 6: D_S (2 features, RAvg) and D_T (18 features, WAvg).

Fig. 7. NMAE measurement for shallow-4 and deep-wide networks.



(a) Scenario 7: D_S (18 features, RAvg) and D_T (2 features, WAvg).



(b) Scenario 8: D_S (2 features, RAvg) and D_T (18 features, WAvg).

Fig. 8. NMAE measurement for shallow-4 and deep-narrow networks.

Each plot shows the transfer gain, transferred model loss, and scratch model loss. Standard deviation, obtained from running the experiments multiple times, is also included.

As can be seen from these plots, transfer gain is larger when transferring from a model with larger feature space (Figure 5a) compared with transferring from a smaller model (Figure 5b). Although transfer gain is limited when transferring from a smaller model to a larger model, generally, a positive transfer gain has been obtained in practically all cases. The gain is reduced with an increasing number of samples in the target domain, which corroborates our previous results [2].

In Figures 6a and 6b, we show results for Scenarios 3 and 4, where heterogeneity is introduced for the features as well as the task, going from *RAvg* in the source domain to *WAvg* in the target domain ($T_S \neq T_T$). A similar observation can been made with the difference that the transfer gain is slightly lower than in Scenarios 1 and 2. Moreover, a small negative transfer gain can be seen in a few cases for a low number of target samples, and when both domains contain only 2 features.

B. Shallow vs. deep neural networks in HTL

In Scenarios 5 and 6, we study the effectiveness of heterogeneous transfer learning using shallow and deep-wide neural network architectures, as stated in Table II. Each scenario is evaluated for three different configurations: shallow-4, deep-wide-6, and deep-wide-10. The source and target models have the same architecture.

In each plot, we calculate the model loss for the scratch and the transferred model, and report the transfer gain. Figure 7a shows the case where the number of features is reduced to 2 in the target domain, from 18 in the source domain, with varying prediction tasks. Figure 7b covers the opposite case. As can be observed, transfer gain increases with the neural-network depth, and a larger gain is obtained with fewer samples. The depth increment also helps to reduce the variance of the transferred model in all cases. More interestingly, transfer gain in deep-wide-10 is still significant when transferring the model from only 2 features (Figure 7b). It is necessary to highlight that the number of features in the source domain highly influences the results obtained in deeper networks. In other words, the model that was trained in the source domain with more features (Figure 7a) has lower variance and reaches lower loss values using fewer training samples, as compared to the model trained with fewer features (Figure 7b).

By having 10 layers, the transferred model loss reaches its minimum even if only few samples are available in the target domain (e.g., 25). It implies that adding more layers does not help in improving the transfer gain any further.

Although the deep-wide neural network options offer several advantages in heterogeneous transfer learning, they come at the cost of a large number of parameters, and thus a longer training time. To address this issue, in Figure 8 corresponding to Scenarios 7 and 8, we investigate the impact of solely increasing the network depth while keeping the number of parameters in the same range as the shallow network. Table II reports the average training time of the shallow, deep-wide, and deep-narrow neural networks over all cases in Figures 7 and 8. As reported in this table, the training time of the deep-narrow neural networks is close to that of the shallow one, while the training time in these networks is shorter than the deep-wide alternatives. Contrastingly, as can be seen in Figure 8, no significant changes in transfer gain have been observed as compared to the deep-wide alternatives in Figure 7.

VI. Related Work

In a seminal paper by Yosinski et al. [5], the authors studied and highlighted the importance of feature transfer, providing insights that neural networks learn general and specific latent features, and that the former can be applied to multiple datasets. Transfer learning is also becoming an increasingly important approach for improving ML performance in dynamic networks, IT, data center, and cloud environments. Examples include our previous works on service performance prediction [2] [3], methods for fault localization in IT infrastructures [11] where transfer learning is incrementally enriching the models, and optimization and configuration of software systems [12] where transfer learning is used as part of a framework to learn from the most relevant sources.

Heterogeneous transfer learning, which is the focus of this paper, has mainly been addressed using *mapping-based* approaches [13]–[15], where features in respective source domains are transformed to a common subspace, thereafter used by a common predictive model. Our work differentiates itself by using a neural-network approach with layer replacement. Recent approaches based on neural networks, proposed in [16], [17], do not investigate feature heterogeneity and the impact of network depth and size.

VII. Conclusion

In this paper, we proposed and evaluated an approach for heterogeneous transfer learning for improved prediction of service performance in networks. The approach enables transfer of feed-forward neural-network models, and thus knowledge, in-between network environments with dissimilar feature sets. We evaluated the approach in multiple heterogeneous scenarios using realistic data sets obtained from a testbed running two different network services under varying load, namely a Video-on-Demand and Key-Value Store service. The obtained results provide empirical evidence showing a positive transfer gain in a majority of cases. For example, the knowledge embedded in a neural network, trained with 18 features in the source domain, can successfully be transferred to a target domain with only 2 input features, and vise versa.

Moreover, choosing an appropriate neural-network depth is crucial to further improve the transfer gain. Deeper networks are embedding more knowledge which results in a higher transfer gain in all scenarios, both in the case of having a large feature set in the source domain and a small feature set in the target domain, and the other way around. Finally, training deeper networks with more features in the source domain has also proven to be an appropriate approach for reducing variance of the transferred models.

Acknowledgment

This research has been supported by the Swedish Governmental Agency for Innovation Systems, VINNOVA, through projects ITEA3 AutoDC and Celtic ANIARA, and Swedish Foundation for Strategic Research, SSF Strategic Mobility.

References

[1] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[2] F. Moradi, R. Stadler, and A. Johnsson, "Performance prediction in dynamic clouds using transfer learning," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 242–250.

[3] H. Larsson, J. Taghia, F. Moradi, and A. Johnsson, "Source selection in transfer learning for improved service performance predictions," in *2021 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2021.

[4] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[5] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *arXiv preprint arXiv:1411.1792*, 2014.

[6] Z. Wang, Z. Dai, B. Póczos, and J. Carbonell, "Characterizing and avoiding negative transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 293–11 302.

[7] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, "A service-agnostic method for predicting service metrics in real time," *International Journal of Network Management*, vol. 28, no. 2, p. e1991, 2018.

[8] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. Long, "Managing flash crowds on the internet," in *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.* IEEE, 2003, pp. 246–249.

[9] J. Ahmed, T. Josefsson, A. Johnsson, C. Flinta, F. Moradi, R. Pasquini, and R. Stadler, "Automated diagnostic of virtualized service performance degradation," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.

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

[11] Y. Shehu and R. Harper, "Towards improved fault localization using transfer learning and language modeling," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2020.

[12] R. Krishna, V. Nair, P. Jamshidi, and T. Menzies, "Whence to learn? transferring knowledge in configurable systems using beetle," *IEEE Transactions on Software Engineering*, 2020.

[13] X. Shi, Q. Liu, W. Fan, S. Y. Philip, and R. Zhu, "Transfer learning on heterogenous feature spaces via spectral transformation," in *2010 IEEE international conference on data mining*. IEEE, 2010, pp. 1049–1054.

[14] J. Zhou, S. Pan, I. Tsang, and Y. Yan, "Hybrid heterogeneous transfer learning through deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, 2014.

[15] L. Duan, D. Xu, and I. Tsang, "Learning with augmented features for heterogeneous domain adaptation," *arXiv preprint arXiv:1206.4660*, 2012.

[16] Z. Xia, L. Wang, W. Qu, J. Zhou, and Y. Gu, "Neural network based deep transfer learning for cross-domain dependency parsing," in *International Conference on Artificial Intelligence and Security*. Springer, 2020.

[17] T. Karb, N. Kühl, R. Hirt, and V. Glivici-Cotruta, "A network-based transfer learning approach to improve sales forecasting of new products," *arXiv preprint arXiv:2005.06978*, 2020.