Exploring Transfer Learning to Reduce Training Overhead of HPC Data in Machine Learning

Tong Liu[1], Shakeel Alibhai[1], Jinzhen Wang[2], Qing Liu[2], Xubin He[1], and Chentao Wu[3]

[1] Department of Computer and Information Sciences, Temple University
[2] Department of Electrical and Computer Engineering, New Jersey Institute of Technology
[3]Department of Computer Science and Engineering, Shanghai Jiao Tong University

***Abstract*—Nowadays, scientific simulations on high-performance computing (HPC) systems can generate large amounts of data (in the scale of terabytes or petabytes) per run. When this huge amount of HPC data is processed by machine learning applications, the training overhead will be significant. Typically, the training process for a neural network can take several hours to complete, if not longer. When machine learning is applied to HPC scientific data, the training time can take several days or even weeks. Transfer learning, an optimization usually used to save training time or achieve better performance, has potential for reducing this large training overhead. In this paper, we apply transfer learning to a machine learning HPC application. We find that transfer learning can reduce training time without, in most cases, significantly increasing the error. This indicates transfer learning can be very useful for working with HPC datasets in machine learning applications.**

***Index Terms*—HPC data, machine learning, transfer learning**

I. Introduction

The training process is an essential step in the machine learning process. In order to make a system learn specific features or patterns, a large training dataset is usually required for training. In addition, in order to achieve high prediction accuracies when testing, training datasets are generally updated through multiple training epochs. Having a large training dataset and many training iterations, however, can put significant pressure on a system's computational resource. As a result, it is not uncommon for the training process of a machine learning task to take hours or even days [1]–[3], if not longer. To resolve this issue, there has been much research into various methods of accelerating the training process [4], [5]. While these approaches may reduce the training time by certain amounts, they are not enough when machine learning is applied to the HPC scientific applications because of the huge amount of data generated through the simulation.

HPC scientific simulations are extremely important, as they allow domain scientists to verify theories and investigate new phenomena on an unprecedented scale [21]. Output from such simulations, however, can frequently reach into the terabytes or even petabytes per run, as HPC simulations record large amounts of data in multiple dimensions (such as spatial and temporal) [6], [7]. When the generated data is analyzed, the huge amount of data processing is likely to be a challenge, especially in machine learning applications. According to the HPC data compression application used in this work, a training process for a binary file of only several megabytes could take hours.

Transfer learning, a machine learning technique, takes a model developed for one task and reuses it as the starting point of a model for a second task [20]. Such an approach aims to reduce the training overhead by reducing the amount of data required for effective training. Transfer learning is commonly used in deep learning applications wherein pre-trained models are used as the staring point of computer vision [8], [9] and natural language processing [10], [11] tasks, as these problems all require vast computational resources and lengthy times to develop neural network models [20]. These are the same challenges experienced in HPC scientific machine learning applications, so transfer learning may also be useful for reducing the lengthy training times for HPC systems. In this paper, we apply transfer learning to a compression autoencoder for lossy data compression as a case study and evaluate its performance with real-world HPC scientific datasets. We find that implementing transfer learning can dramatically reduce the training time without, in most cases, netagively impacting performance. For datasets in the same domain, it is very common to see roughly equal, if not improved, performance when using transfer learning, while for cross-domain datasets, we see such results in at least some of the cases.

II. Background

A. Transfer Learning

Traditionally, machine learning algorithms [12]–[14] make predictions on future data by utilizing models trained on previously collected data. In order to have an accurate predictor, it is important for a model to be well-trained. However, there exist several issues relating to the training step. These problems include not having enough labelled data for training or having very high training overhead. In order to alleviate these problems, semi-supervised classification [15]–[18] has been proposed. This technique that makes use of a large amount of unlabelled data and a small amount of labelled data for training. Nevertheless, most of these work will assume that the distributions of the labelled and unlabelled data are the same. In contrast, transfer learning, allows the domains, tasks, and distributions used in training and testing to be different, which makes some impossible training process practical.

In general, transfer learning aims to extract knowledge from

978-1-7281-4409-2/19/$31.00

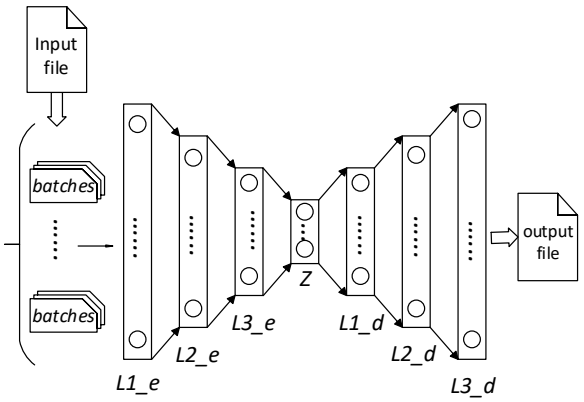


Fig. 1: The seven-layer compression autoencoder prototype.

one or more source tasks and apply it to a target task, which is usually in a different domain than the source tasks. As a formal definition [19], assume that we have a source domain D_s, a learning task T_s, a target domain D_t, and learning task T_t. By implementing transfer learning, we aim to improve the learning of the target predictive function $f_t(\cdot)$ in D_t using the knowledge in D_s and T_s where $D_s \neq D_t$ or $T_s \neq T_t$.

In the above definition, the condition $D_s \neq D_t$ implies that either the training data domains or the data distributions of the source and target should be different. Similarly, $T_s \neq T_t$ implies that either the testing datasets or the prediction functions of the source and target should differ. It should be noted that when $D_s = D_t$ and $T_s = T_t$ (meaning the domains and learning tasks of the source and target domains are the same), the learning problem becomes a traditional machine learning problem.

B. HPC lossy data compression with compression autoencoder

In this paper, we use an HPC lossy compressor application as a case study for transfer learning with HPC data. This application uses a compression autoencoder (CAE) to conduct dimensional reduction and thus achieve data compression.

The autoencoder compression prototype is shown in Figure 1. The autoencoder has seven layers with three layers $L1_e, L2_e, L3_e$ in the encoder part, three layers $L1_d, L2_d, L3_d$ in the decoder part, and one code layer Z. Before the input file I (which contains the scientific data) enters the neural network, it is divided into several batches $b_i \in I$. After the input is divided into batches, each batch b_i will contain a part of the original scientific data and then go into the input layer $L1_e$ for training. When a batch enters the input layer, each element of the original scientific data becomes one neuron in the layer. In the encoder part, the number of neurons decreases as the layer goes from the input layer $L1_e$ to the code layer Z. Each layer in the encoder part has a weight matrix and bias vector that are used to accomplish dimension reduction. After three layers of compression, the information stored in $L1_e$ is represented in layer Z with significantly fewer neurons. The decoder is similar to the encoder in that each layer has a weight matrix and bias vector. The information in the Z layer goes through the three decoder layers and is then written to the output file. If the whole autoencoder is regarded as a compressor, then layer $L1_e$ is the original file, layer Z is the compressed file, layer $L3_d$ is the decompressed file, and the encoder and decoder represent the compression and decompression, respectively.

III. Observation and Experiment Setup

A. Training time observation

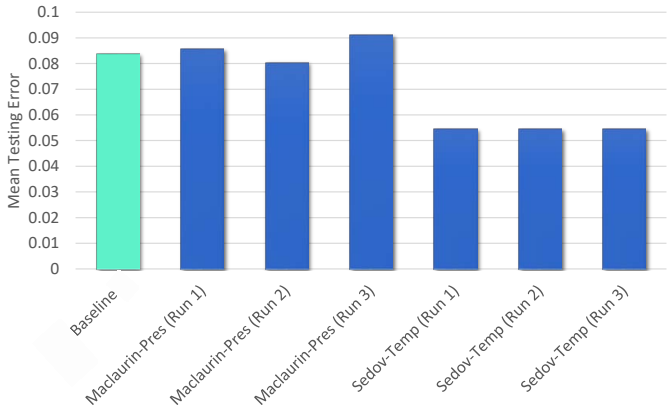
In our preliminary experiments, we find that a file with around 400,000 double-precision floating-point numbers (around 3.2 MB) will need approximately two hours for training with 25,000 epochs. In order to see how training time can increase as the size of the training dataset increases, we perform experiments on a series of datasets.[1] The datasets we use for testing come form open-source HPC scientific data benchmarks, such as NEK5000 [38], MCERI [39], GROMACS [40], and FLASH [41]. These experiments are conducted with 2,500 epochs on a system running Ubuntu 16.04.5 LTS with an Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz and 32 GB of memory (clock: 2133MHz). The results are presented in Table I.

TABLE I: Training time for various datasets with 2,500 epochs.

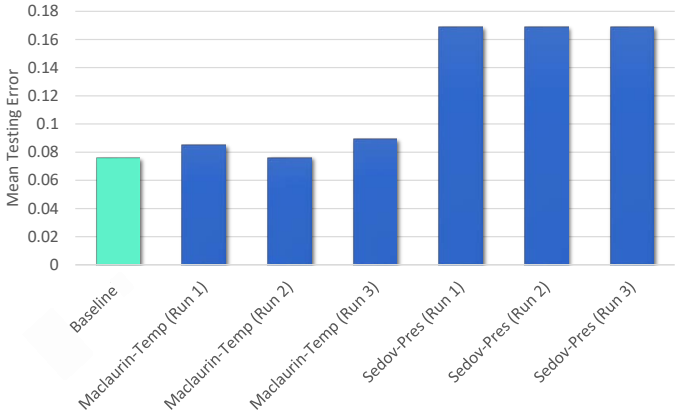
Dataset	Number of Elements	Approximate Training Time
bump_training.bin	25,000	94.185 seconds
yf17_pres_training.bin	48,552	170.118 seconds
eddy_training.bin	135,000	467.876 seconds
md_training.bin	300,000	1057.617 seconds
Randomly-Generated Dataset A	500,000	1703.223 seconds
Randomly-Generated Dataset B	1,000,000	3479.543 seconds
Randomly-Generated Dataset C	5,000,000	17175.178 seconds
Randomly-Generated Dataset D	10,000,000	34684.248 seconds

As seen from the results, the training time increases as the size of the training dataset increases in a roughly linear fashion. In addition, as noted above, these results are for 2,500 epochs. To achieve greater accuracy, a higher number of training epochs may be needed. During the experiments, we find that 25,000 is a suitable number of epochs, and having ten times the number of epochs would increase the training time by approximately ten times. The problem of extensive training time is further intensified by the fact that HPC datasets can be terabytes or even petabytes in size. Training times for those datasets would be much higher than even the training time for Randomly-Generated Dataset D, which has 10 million data points and yet is only around 77 MB.

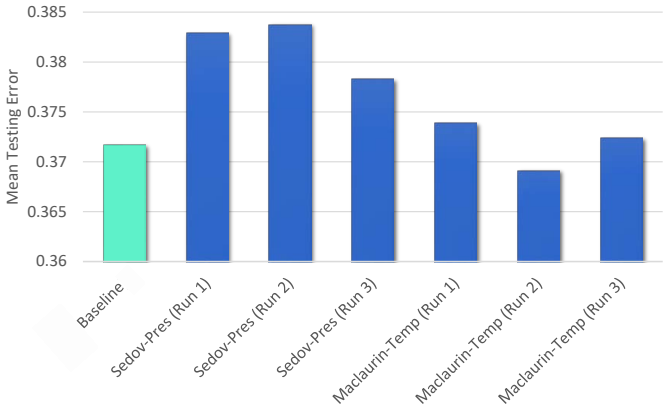
[1]With the exception of the randomly-generated datasets, all datasets used in this paper are provided at the link in the Section VI.



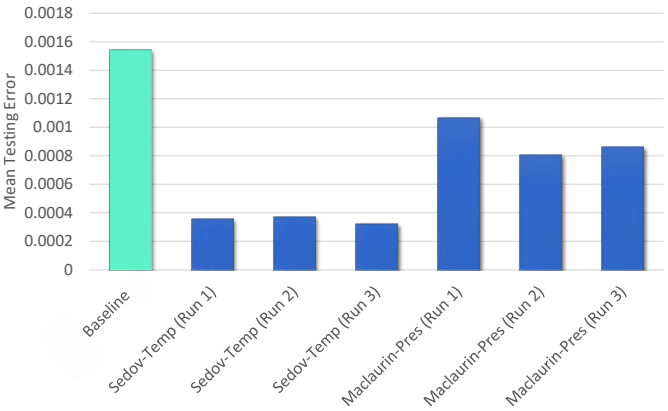
(a) Mean testing errors upon compressing Maclaurin-Temp



(b) Mean testing errors upon compressing Maclaurin-Pres



(c) Mean testing errors upon compressing Sedov-Temp



(d) Mean testing errors upon compressing Sedov-Pres

Fig. 2: Preliminary series of experiments utilizing the concept of transfer learning. The x-axis indicates which dataset, if any, was used as the original training dataset. Note that each simulation (Maclaurin-Temp, Maclaurin-Pres, Sedov-Temp, and Sedov-Pres) has two separate datasets, one that is used for training and another that is used for testing (ie. compression).

B. Experiment setup

In order to attempt to reduce training time, we develop a basic implementation of transfer learning in a compression autoencoder. The purpose of training is to generate weights and biases that can be utilized later. In the compression autoencoder, when training occurs without transfer learning, the weights are initially set to random values from a normal distribution and the biases are initialized to zero. These weights and biases are then optimized while the autoencoder runs through consecutive epochs. Assume that we train one dataset, dataset A, and obtain weights and biases w_A and b_A. These weights and biases are optimized for dataset A; they can be used on any datasets similar to A, such as A', but simply using these on a different dataset, dataset B, will not necessarily yield acceptable results.

Now assume we want to train dataset B. Without transfer learning, we would again start with randomly-initialized weights and zero-initialized biases. We would then go through multiple epochs, and in each epoch, we would go through every data point in dataset B. As previously noted, this could be very time-consuming if many epochs are needed or if dataset B is very large—and it is very common for both these conditions to be met. Thus, in order to reduce the time to train dataset B, we set the initial values of w_B and b_B to be based on w_A and b_A instead of setting them to random values and zeros, respectively. However, it is possible that the features of dataset B could be very different from those of dataset A, meaning that w_A and b_A may not be very applicable for dataset B. Thus, in order to reduce the impact of dataset A on w_B and b_B, we multiply the values of w_A and b_A by 0.7.[2] This means that w_B is initialized to $w_A * 0.7$ and b_B is initialized to $b_A * 0.7$. Since the possible features of the HPC dataset are already represented by w_A and b_A, we do not need to use the entirety of dataset B for training; instead, we can save valuable time by only training on a portion of dataset B. As a result, we take p data points of dataset B for training, rather than the entire dataset. In our simple implementation, we generally set p as 30% of the size of dataset A. (If the size of dataset B is smaller than p, then we use the entirety of dataset B

[2]In this case, 0.7 serves as a starting point for our basic implementation of transfer learning. Future work could determine what the optimal number to multiply the weights and biases by is or how to find such a number.

for training, but this situation is relatively uncommon.) In our experiments, we then optimize w_B and b_B for the first p data points of dataset B for 2,500 epochs. Table I indicates that smaller datasets need less time for training, and since only a portion of dataset B (rather than the full dataset) is used for training, we can save a significant amount of time.

IV. Evaluation

One major question that can arise from the implementation discussed in Section III.B is whether the resulting weights and biases, w_B and b_B, are still effective or whether they perform poorly when they are used. In order to gain insight into whether the weights and biases are still usable (without a significant degradation of accuracy) after this implementation, we performed a series of tests. We take four datasets (Maclaurin-Pres, Maclaurin-Temp, Sedov-Pres, and Sedov-Temp) and run them in the compression autoencoder. The baseline performance is the testing error when transfer learning is not used (ie. dataset X is trained and weights and biases w_x and b_x are generated; those weights and biases are then tested by being used to compress dataset X'; the compression error (ie. the testing error) is recorded); these are the leftmost bars in each graph (in light green). For each of the four datasets, we choose two datasets to compare it with using transfer learning. For example, one of the datasets with Sedov-Pres is Sedov-Temp (Figure 2d). To apply transfer learning, we train Sedov-Temp to obtain weights and biases w_t and b_t. We then multiply these weights and biases by 0.7 and optimize them using a portion of Sedov-Pres, as explained in Section III.B. This test is performed three times in order to ensure that the results are not simply due to randomness. The results are in Figure 2. Note that lower bars (ie. lower errors) signify better performance.

From these four tests, we see that this method is promising. For each of the four datasets in this experiment, not only is it possible to obtain weights and biases with similar performance to those generated without transfer learning, but it is possible to obtain weights and biases that have better performance (ie. lower errors) than those generated previously. (It is important to note, however, that the performance does not improve in all cases.)

Finding 1: Using transfer learning to train a dataset based on previously-generated weights and biases can decrease the training time and frequently has acceptable performance, thus indicating that transfer learning is a promising technique to reduce the training overhead for HPC machine learning applications.

We observe that, in these experiments, whenever Sedov-Temp is used as the original dataset, the training error is reduced. As a result, we conduct another series of experiments wherein we take Sedov-Temp as the original training dataset to generate weights and biases w_t and b_t. We use these weights and biases as the starting point for training a variety of other datasets. We then use those weights and biases to compress similar datasets. (For example, we may generate weights and biases from Sedov-Temp and then use them to generate new weights and biases, w_p and b_p, on a portion of Sedov-Pres. There are two Sedov-Pres datasets, one for training and one for testing. A portion of the training dataset is used for obtaining w_p and b_p, and the testing dataset is compressed using these weights and biases.) We compare the compression error using this method to the error without using transfer learning. The results, indicating the performance improvement, are shown in Table II.

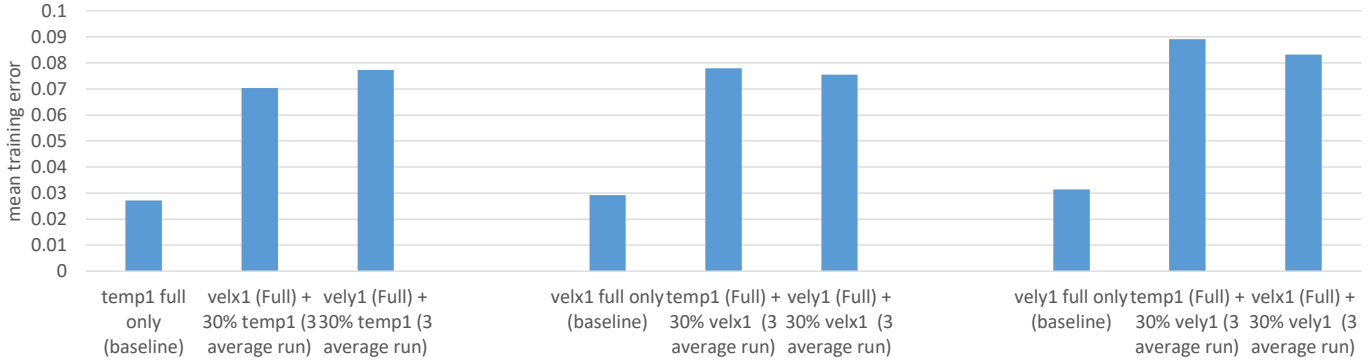
TABLE II: Transfer learning results with Sedov-Temp as the original training dataset.

Dataset	**Number of Elements**	**Approximate Performance Improvement When Sedov-Temp is Used**
Sedov-Pres	39,072	442.9%
Maclaurin-Temp	133,376	153.3%
YF17-Temp	48,552	140.9%
S3DP	97,020	132.5%
Astro	32,768	123.2%
YF17-Pres	48,552	107.4%
Blast2	289,440	98.4%
Fish	32,768	93.2%
Eddy	135,000	84.8%
Md-Seg	300,000	76.1%
2D-Annulus	181,890	50.4%
Swept	77,180	40.7%

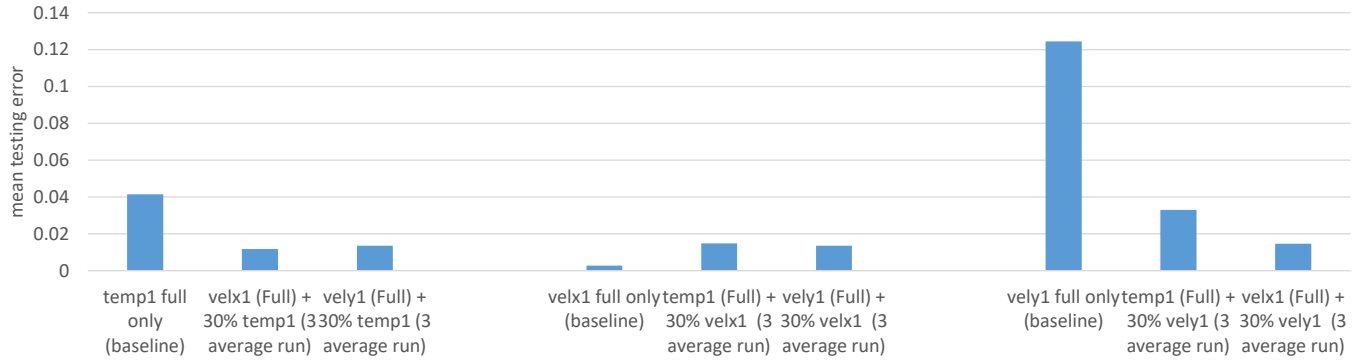
From this table, we can see that Sedov-Pres experiences vastly improved performance when Sedov-Temp is used as the original training dataset. Several other datasets also see performance improvements, and two of them, Blast2 and Fish, only see slight reductions in performance. Four datasets, however, experience more significant performance reductions.

From these results, we can conclude that using transfer learning is promising. We are able to significantly reduce the training time, which can be extremely important for large datasets. In many cases, we obtain the added benefit of having improved accuracy. While the accuracy does decrease in some cases, this could be solved by using a different dataset as the original training dataset or using different parameters instead of the aforementioned 0.7 and 30%. Further research into these cross-domain trends is potential future work.

While using cross-domain datasets for transfer learning may have some ambiguity, we find that using transfer learning with datasets within the same domain may have more consistent performance. To explore transfer learning performance within the same simulation, we choose three datasets from three simulations, *2d_annulus*, *maclaurin*, and *sedov*, and generate different metrics within each simulation. For *2d_annulus*, we collect temperature data (*temp*), velocity data on the x-axis (*velx*), and velocity data on the y-axis (*vely*), each for 121,260 timesteps. For *maclaurin*, we collect temperature (*temp*) and pressure data (*pres*) for 266,752 timesteps. For *sedov*, we collect temperature (*temp*) and pressure data (*pres*) for 78,144 timesteps. Each of these metrics are divided into two equal parts, one for training and another for testing.

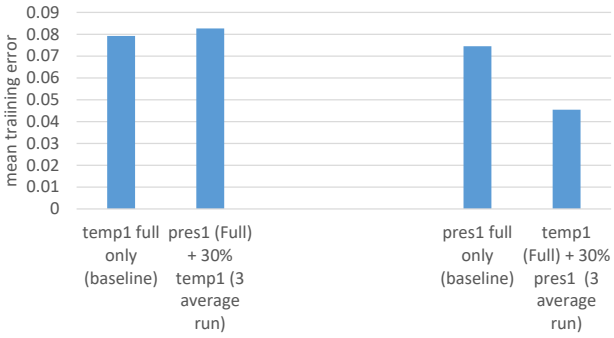


(a) Mean training errors for three metrics from the *2d_annulus* simulation.

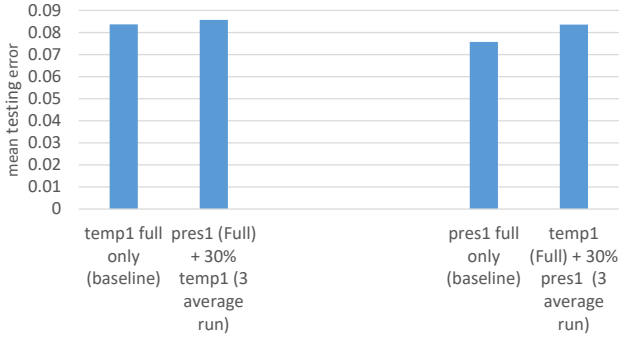


(b) Mean testing errors for three metrics from the *2d_annulus* simulation.

Fig. 3: Mean training and testing errors for three metrics, *temp*, *velx*, and *vely*, from the *2d_annulus* HPC simulation.



(a) Mean training errors for two metrics from the *maclaurin* simulation.

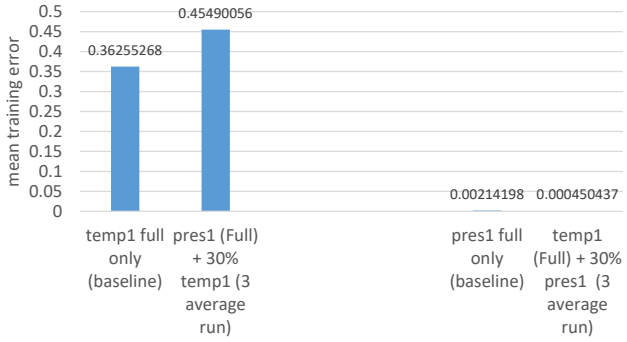


(b) Mean testing errors for two metrics from the *maclaurin* simulation.

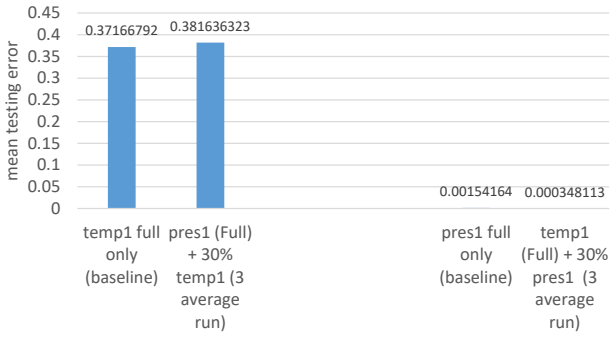
Fig. 4: Mean training and testing errors for two metrics, *temp* and *pres*, from the *maclaurin* HPC simulation.

Figures 3, 4, and 5 show the results for both training and testing on these metrics.[3] For each simulation, we first conduct traditional machine learning; this serves as the baseline performance. We then use transfer learning from the other metrics in the same simulation on that metric to compare the performance of baseline and transfer learning. Transfer learning was impelemented in the same way as in the previous experiments. From these results, we can see that if we use transfer learning within one simulation, the,n in most of the cases, the testing results with transfer learning are similar to the baseline performance, if not better. In certain cases, such as with *velx*, the transfer learning results are slightly worse than the baseline, but they may still remain roughly around the same level as the baseline with low error (less than 0.02 in that case). Note that worse training performance does not necessarily mean that transfer learning

[3]The testing results are the results used to determine performance. Having better or worse training results does not necessarily indicate that transfer learning performs better or worse than the baseline; this determination is made using the testing results.



(a) Mean training errors for two metrics from the *sedov* simulation.



(b) Mean testing errors for two metrics from the *sedov* simulation.

Fig. 5: Mean training and testing errors for two metrics, *temp* and *pres*, from the *sedov* HPC simulation.

performs worse, as the testing performance is the metric that determins how transfer learning performs in comparison to the baseline. In these experiments, the training time is reduced by around 70% while still maintaining roughly the same (if not better) level of prediction error. This matches the intuition that metrics within the same simulation usually share similar distributions, even though they may be different. For example, when running astrometeorology simulations, locations with higher temperatures will also have higher pressures; this indicates that metrics inside one simulation may frequently be correlated.

Finding 2: When implementing transfer learning, if the original training dataset and the second training dataset are in different simulations, then the resulting weights and biases often perform well, but occassionally perform very poorly. When both training datasets are from the same simulation, however, using transfer learning is very likely to produce results similar to, if not better than, the baseline performance.

In addition, we see another interesting observation: in some of the cases, using transfer learning leads to higher training error but lower testing error. This could be because, in those cases, the model experiences overfitting, a common problem where training leads to the weights and biases being too well-fit to the training data and not generalized enough to work as well on similar datasets [22]. This could indicate that transfer learning can, in some cases, serve as a foil against overfitting.

V. Related Work

Much research has been performed to accelerate training time by minimizing the training overhead. For example, Osuna et al. [23] present a decomposition algorithm that is guaranteed to solve the quadratic programming problem to improve training performance. Joachims et al. [24] analyze particular properties of learning with text data and explore the use of support vector machines to reduce training time. Work has also been performed by Microsoft [25] that uses sequential minimal optimization to avoid using a time-consuming numerical QP optimization as an inner loop.

Transfer learning, as a hot topic that aims to extract knowledge from one domain to another, has been applied to multiple machine learning scenarios recently. Transfer learning research can be generally divided into four categories based on the way it is implemented [19]. 1) Instance transfer [26]–[28], in which some labelled data in the source domain is re-weighted before being used in the target domain. 2) Feature-representation-transfer [29]–[31], which finds a feature representation that reduces differences between the source and target domains. 3) Parameter-transfer [32]–[34], which discovers shared parameters between the source and target domain models that can benefit from transfer learning. 4) Relational-knowledge-transfer [35]–[37], which builds a mapping of relational knowledge between the source and target domains. Our transfer learning scheme in this paper is similar to the first category, as we take advantage of external knowledge from the source domain data's weights and biases by applying a weighted version of them to the target domain that we want to test.

VI. Conclusion

In this paper, we explore the possibility of using transfer learning on HPC data by using a compression autoencoder (CAE) as a case study. We implement transfer learning in a CAE; this generally leads to a vast reduction in training time. We find that when transfer learning is applied to datasets within the same simulation, it is highly likely to achieve similar or even better performance than traditional machine learning. In addition, we run a series of experiments with a fixed dataset for the original transfer learning and 12 datasets that use the results of that transfer learning to generate their own weights and biases. In the majority of these 12 cases, the performance is either very similar to or much better than the performance of traditional machine learning, indicating that transer learning can even be possible for cross-domain datasets. Our future work could include exploring ways of determining which data features make datasets suitable for cross-domain transfer learning and looking into ways of

further improving the performance of transfer learning. As transfer learning can reduce training time without, in most cases, significantly increasing the error, transfer learning can be very useful for working with HPC datasets in machine learning applications.

The source code for the compression autoencoder (both the original version and the version with transfer learning) and the HPC datasets used in this paper are available at https://github.com/tobivcu/autoencoder.

The work performed is partially sponsored by the U.S. National Science Foundation grants CCF-1813081, CNS-1702474, CCF-1718297, and CCF-1812861. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] S. Shen, et al. Minimum risk training for neural machine translation. arXiv preprint arXiv:1512.02433 (2015).

[2] P. Stone, et al. Keepaway soccer: From machine learning testbed to benchmark. Robot Soccer World Cup. Springer, Berlin, Heidelberg, 2005.

[3] J. Snoek, L. Hugo, and P. Ryan Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems. 2012.

[4] M. Di, and M. Er. A survey of machine learning in wireless sensor netoworks from networking and application perspectives. 2007 6th international conference on information, communications & signal processing. IEEE, 2007.

[5] B. Catanzaro, S. Narayanan, and K. Kurt. Fast support vector machine training and classification on graphics processors. Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[6] T. Lu, et al. Canopus: A paradigm shift towards elastic extreme-scale data analytics on hpc storage. 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2017.

[7] D. Tao, et al. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2017.

[8] C. Lampert, N. Hannes, and H. Stefan. Learning to detect unseen object classes by between-class attribute transfer. 2009 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2009.

[9] H. Shin, et al. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE transactions on medical imaging 35.5 (2016): 1285-1298.

[10] A. Conneau, et al. Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364 (2017).

[11] R. Collobert, and W. Jason. A unified architecture for natural language processing: Deep neural networks with multitask learning. Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[12] X. Yin, J. Han, J. Yang, and P. S. Yu, Efficient classification across multiple database relations: A crossmine approach, IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 6, pp. 770783, 2006.

[13] L. I. Kuncheva and J. J. Rodrguez, Classifier ensembles with a random linear oracle, IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 4, pp. 500508, 2007.

[14] E. Baralis, S. Chiusano, and P. Garza, A lazy approach to associative classification, IEEE Transactions on Knowledge and Data Engineering, vol. 20, no. 2, pp. 156171, 2008.

[15] X. Zhu, Semi-supervised learning literature survey, University of WisconsinMadison, Tech. Rep. 1530, 2006.

[16] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell, Text classification from labeled and unlabeled documents using em, Machine Learning, vol. 39, no. 2-3, pp. 103134, 2000.

[17] A. Blum and T. Mitchell, Combining labeled and unlabeled data with co-training, in Proceedings of the Eleventh Annual Conference on Computational Learning Theory, 1998, pp. 92100.

[18] T. Joachims, Transductive inference for text classification using support vector machines, in Proceedings of Sixteenth International Conference on Machine Learning, 1999, pp. 825830.

[19] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 13451359.

[20] A Gentle Introduction to Transfer Learning for Deep Learning. https://securityglobal24h.com/a-gentle-introduction-to-transfer-learning-for-deep-learning/machine-learning/Information-Security-latest-Hacking-News-Cyber-Security-Network-Security. Accessed May 7, 2019.

[21] T. Lu, et al. Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data in IEEE International Parallel and Distributed Processing Symposium, 2018.

[22] T. Dietterich, "Overfitting and undercomputing in machine learning" in ACM computing surveys, 1995.

[23] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines, IEEE Workshop on Neural Networks and Signal Processing, Amelia Island,IEEE Press, 1997: 276- 285.

[24] T. Joachims. "Text categorization with support vector machines: Learning with many relevant features." European conference on machine learning. Springer, Berlin, Heidelberg, 1998.

[25] J. Platt. "Sequential minimal optimization: A fast algorithm for training support vector machines." (1998).

[26] W. Dai, Q. Yang, G. Xue, and Y. Yu, Boosting for transfer learning, in Proceedings of the 24th International Conference on Machine Learning, Corvalis, Oregon, USA, June 2007, pp. 193200.

[27] W. Dai, G. Xue, Q. Yang, and Y. Yu, Transferring naive bayes classifiers for text classification, in Proceedings of the 22rd AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, July 2007, pp. 540545.

[28] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, Dataset Shift in Machine Learning. The MIT Press, 2009.

[29] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, Self-taught learning: Transfer learning from unlabeled data, in Proceedings of the 24th International Conference on Machine Learning, Corvalis, Oregon, USA, June 2007, pp. 759766.

[30] W. Dai, G. Xue, Q. Yang, and Y. Yu, Co-clustering based classification for out-of-domain documents, in Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 2007.

[31] R. K. Ando and T. Zhang, A high-performance semi-supervised learning method for text chunking, in Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Morristown, NJ, USA: Association for Computational Linguistics, 2005, pp. 19.

[32] N. D. Lawrence and J. C. Platt, Learning to learn with the informative vector machine, in Proceedings of the 21st International Conference on Machine Learning. Banff, Alberta, Canada: ACM, July 2004.

[33] E. Bonilla, K. M. Chai, and C. Williams, Multi-task gaussian process prediction, in Proceedings of the 20th Annual Conference on Neural Information Processing Systems. Cambridge, MA: MIT Press, 2008, pp. 153160.

[34] A. Schwaighofer, V. Tresp, and K. Yu, Learning gaussian process kernels via hierarchical bayes, in Proceedings of the 17th Annual Conference on Neural Information Processing Systems. Cambridge, MA: MIT Press, 2005, pp. 12091216.

[35] L. Mihalkova, T. Huynh, and R. J. Mooney, Mapping and revising markov logic networks for transfer learning, in Proceedings of the 22nd AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, July 2007, pp. 608614.

[36] L. Mihalkova and R. J. Mooney, Transfer learning by mapping with minimal target data, in Proceedings of the AAAI-2008 Workshop on Transfer Learning for Complex Tasks, Chicago, Illinois, USA, July 2008.

[37] J. Davis and P. Domingos, Deep transfer via second-order markov logic, in Proceedings of the AAAI-2008 Workshop on Transfer Learning for Complex Tasks, Chicago, Illinois, USA, July 2008.

[38] Nek5000 User Guide. https://nek5000.mcs.anl.gov/files/2015/09/NEK_doc.pdf.

[39] Trace3D, 3D streamline simulator. http://www.pe.tamu.edu/mceri/software.html.

[40] GROMACS, molecular dynamics performance simulator. http://www.gromacs.org/About_Gromacs/Benchmarks.

[41] ASCF Center. FLASH Users Guide. http://flash.uchicago.edu/site/flashcode/user_support/.