Using Multi-Resolution Data to Accelerate Neural Network Training in Scientific Applications

Kewei Wang*, Sunwoo Lee‡, Jan Balewski†, Alex Sim†, Peter Nugent†,
Ankit Agrawal*, Alok Choudhary*, Kesheng Wu†, and Wei-keng Liao*

*ECE Department, Northwestern University
{kwf5687, ankitag, choudhar, wkliao}@ece.northwestern.edu
†Lawrence Berkeley National Laboratory
{balewski, asim, penugent, kwu}@lbl.gov
‡ University of Southern California
sunwool@usc.edu

***Abstract*—Neural networks are powerful solutions to many scientific applications; however, they usually suffer from long model training times due to the typical data size and model size being large. Research has been focused on developing numerical optimization algorithms and parallel processing to reduce the training time. In this work, we propose a multi-resolution strategy that can reduce the training time by training the model with the reduced-resolution data samples at the beginning and later switching to the original resolution data samples. This strategy is motivated by the fact that many scientific applications run faster when using a coarse version of the problem, for example, data whose resolution is reduced statistically. When applying the idea to neural network training, coarse data can have a similar effect on the learning curves at the early stage as the dense data but requires less time. Once the curves no longer improve significantly, our strategy switches to using the data in original resolution. We use two real-world scientific applications, CosmoFlow and DeepCAM, to evaluate the proposed mixed-resolution training strategy. Our experiment results demonstrate that the proposed training strategy effectively reduces the end-to-end training time while achieving a comparable accuracy to that of the training only with the original data. While maintaining the same model accuracy, our multi-resolution training strategy reduces the end-to-end training time up to 30% and 23% for CosmoFlow and DeepCAM, respectively.**

***Index Terms*—Deep Learning, Transfer Learning, Multi-resolution Data**

I. Introduction

Many scientific applications have successfully used deep learning to solve their data analysis tasks [1]–[5]. One common challenge in deep learning is to shorten the long training time for the neural networks, which can be hours or days. To have a reasonable training time, researchers have proposed various techniques. For example, there are many approaches for speeding up the training procedure by improving scalability, such as large-batch training [6], [7], exploiting different forms of parallelism [8], asynchronous training [9], reducing communication during training [10], [11], and so on. Other approaches focus on the statistical efficiency of optimization algorithms to reduce the number of training iterations, such as AdaGrad [12], Adam [13], AdamW [14] and variance-reduced SGD [15], [16]. In this work, we approach the issue of long training time from a different angle. We exploit a physics principle to enable an effective transfer learning procedure to provide an alternative strategy.

In this paper, we propose a multi-resolution training strategy (MRT) that reduces both computation and communication time when training the neural network model in parallel. The strategy trains the model using reduced-resolution data samples first and later switches to the original-resolution data. Training the model with coarse-resolution data takes a shorter time than the original data because each training sample size is reduced, resulting in less computation cost. Once the loss curve stops improving, we switch back to use the original resolution data to continue the training. At the end when the training converges, a comparable accuracy can be achieved as if the model is trained entirely using the original data. We take the continuity in scientific problems to generate coarser versions of the input data and demonstrate that simple strategies for reducing data resolution could work well. To construct a model for training the coarse data, we present a working choice of sub-network transfer which requires a small adjustment on the model architectures. We also propose a switching mechanism that triggers the switch based on the changes of training loss in a given time window to decide when to switch from the coarse to the dense data.

Our training strategy is motivated by multigrid strategies in scientific computing [17], [18]. The data from scientific problems often are the discretization of some physical quantities, such as temperature and pressure from an atmospheric application, DeepCAM [19], dark matter mass distribution in CosmoFlow [3]. In these cases, the physical quantity could be discretized at different resolutions; a coarse resolution would lead to smaller arrays while a denser resolution leads to larger arrays. Various multigrid strategies have been proposed to take advantage of the multiple resolutions of the same physics quantities [17], [18]. The main idea behind these strategies is that a scientific problem could be solved with less computational effort at a coarse resolution, though the solution might have more uncertainty. For iterative methods, such coarse data can potentially be used at the beginning of

the iterations until the intermediate solution reaches a similar value as the one when using the dense data only. The time to reach such a point is expected to be smaller. The same idea can be applied to neural network training if we can transfer the model trained with coarse data at the beginning to train with dense data.

We evaluate the multi-resolution training strategy using two real-world scientific applications, CosmoFlow and DeepCAM, from the MLPerf HPC v0.7 training benchmark suite [20], [21]. Both applications have large multi-channel datasets and suffer from expensive computational time during the training. Our experiments were carried out on two supercomputers, Summit at Oak Ridge National Laboratory (ORNL) and Cori at National Energy Research Scientific Computing (NERSC). The goal of the multi-resolution strategy is to reduce the end-to-end model training time while keeping the same validation accuracy as the model trained with the original data. We empirically verify that our training strategy can effectively transfer knowledge from the low-resolution data through network-based transfer learning. The transferred knowledge from the pre-trained model successfully boosts the training efficiency of the original model. Compared to training on the original model, we observe an end-to-end training time improvement of up to 30% and 23% for CosmoFlow and DeepCAM without losing accuracy. In addition, we present the scaling performance results of running the training in parallel on 32 to 128 GPUs, with the timing breakdown of I/O, training, and inter-process communication. We also study the impact of different switching points from coarse to dense data on the total training time. Our experimental results demonstrate that the proposed strategy can successfully reduce end-to-end training time without losing model accuracy.

II. Background and Related Work

A. Continuity in Scientific Applications

In many scientific applications, the data are usually physical quantities discretized in the space or time domain. Thus, there is inherent continuity among the data elements. Consequently, it is possible to generate a coarser version of the datasets by simply sampling fewer numbers of data elements or by taking averages of neighbor elements. In this work, we consider a few representative coarsening schemes.

In scientific computing, there are many other strategies to make use of this continuity. For example, Suisalu et al. [18] solve cosmology problems using multigrid methods. A similar multigrid method is also used to solve climate modeling problems [17]. Note that the multigrid procedures have a strong mathematical foundation. These previous works show effective use-cases of multigrid methods. We adopt this core idea to large-scale deep learning to accelerate neural network training.

B. Transfer Learning

Transfer learning techniques have shown to be effective in many deep learning applications [22]–[24]. Transfer learning techniques are usually designed to get useful knowledge transferred across different domains. When using neural networks for transfer learning, they are first pre-trained in the source domain, and then either a part of the model or the whole model is re-trained on the destination domain datasets. For example, ImageNet is usually used in computer vision tasks to pre-train the model before further fine-tuning for the downstream tasks [25], [26]. When the target dataset is small, fine-tuning the whole model might cause overfitting [27]. Thus, the few input side layers can be frozen while the rest are fine-tuned. Various works have studied which and how many layers should be frozen during fine-tuning [27], [28]. In this paper, we borrow the neural network-based transfer learning approaches to transfer knowledge from the low-resolution task to the original task for the purpose of speeding up overall training.

C. Synchronous SGD with Data Parallelism

Many machine learning applications use stochastic gradient descent or its variants to solve domain-specific optimization problems. Mini-batch SGD iteratively utilizes a random subset of training data points to compute the gradients for adjusting model parameters. To reduce the training time, researchers have proposed many parallel training algorithms. Synchronous SGD representing the synchronous-parallel version of mini-batch SGD is the most conventional algorithm. With data parallelism, each mini-batch is evenly distributed to workers. Each worker processes the assigned data to compute the gradients independently. Then, the gradients are averaged across all the workers using inter-process communication. We use synchronous SGD with data parallelism that provides the optimal statistical efficiency compared to other communication-efficient algorithms that can potentially harm the accuracy.

D. CosmoFlow

CosmoFlow is a project that develops a deep learning tool for analyzing cosmology data. It is included in the MLPerf HPC Training benchmark [20], an industry-standard performance benchmark for evaluating Machine Learning performance on large-scale HPC systems. Mathuriya et al. proposed adopting a 3D convolutional neural network to estimate the initial condition of the universe based on the 3D simulations of the distribution of matter [3], [21]. This application incorporates large multi-dimensional data samples containing 3D cubes of size 128^3 with four redshift channels. The large size of the CosmoFlow dataset makes it computationally expensive to train over many iterations.

E. DeepCAM

The Community Atmospheric Model is a global atmosphere model for simulating long-term climate trends in the earth's atmosphere [19]. One way to validate such a simulation is to extract critical atmospheric events such as atmospheric rivers and tropical cyclones. DeepCAM is a test benchmark for using deep learning to identify such events from a CAM5 simulation dataset [19]. The data samples in the test dataset are defined as 2D meshes of the earth's surface, and labels

are provided for each mesh point when an event of interest is present. This test problem is also included in the MLPerf HPC Training benchmark. The dataset of DeepCAM contains meshes of size 768 x 1152 with 16 variables at each mesh point. With such massive data volume and a large encoder-decoder segmentation architecture, the application DeepCAM is computationally challenging.

III. Multi-resolution training

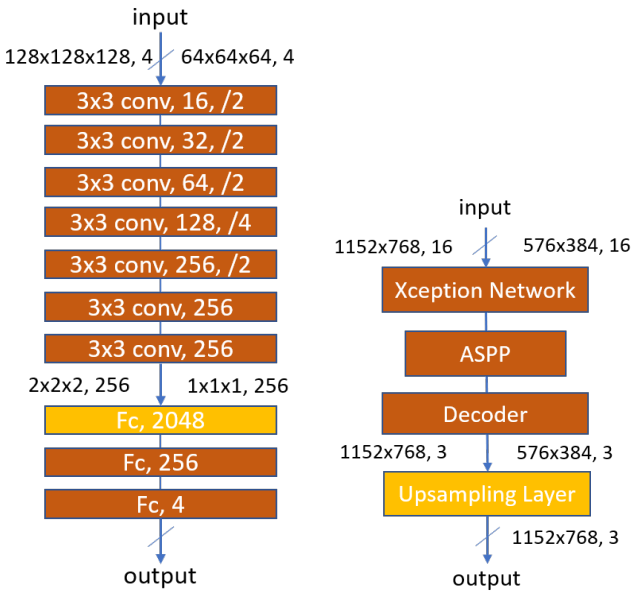
The proposed multi-resolution training strategy (MRT) draws inspiration from the multigrid solution strategies; that is, solving the coarser version of a problem could help in solving the denser version of the same problem. In addition, there is an observation that when neural networks are trained on images, the first layer features resemble each other regardless of the exact dataset [29]. For features transferred from different tasks, it is shown that learned features are better than random initialization, even for distant tasks [27]. Thus, we expect that the transferred weights from the model trained on a lower resolution dataset of the same task can be beneficial to the training on the dense dataset.

Given a scientific dataset with large data samples, training a neural network can be time-consuming. The training time can be largely reduced by reducing the size of training samples by reducing their data resolution. However, if we train the neural network with low-resolution data only, the generalization performance will be affected, causing higher validation loss and lower accuracy. To efficiently utilize the low-resolution data to train the neural network while not hurting the model's generalization performance, we propose adopting a two-stage training procedure.

Training procedure – Given a deep learning model $\mathcal{M}$, i.e., a function $\mathcal{M}(x_i, y_i, w) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ mapping n-dimensional input samples X to m-dimensional labels Y with parameters (weights) w, we use the original dataset with samples X and labels Y as the dense dataset. Firstly, we preprocess the input samples X into coarse samples X_c and labels Y into Y_c as the coarse dataset. Then, in the first training stage, we construct the coarse model $\mathcal{M}_c$ based on $\mathcal{M}$ and train the neural network with the coarse dataset, X_c and Y_c, for T_c iterations. After having the coarse model $\mathcal{M}_c$ pre-trained with the coarse set X_c, we pass the weights of partial layers from $\mathcal{M}_c$ to initialize the original model $\mathcal{M}$, the dense model. The rest layers in the dense model are initialized with random weights. Then, in the second training stage, we employ the dense dataset X and Y to fine-tune the dense model $\mathcal{M}$ for T iterations.

A. Creating Coarse Training Dataset

Given a set of training samples, we reduce their resolutions uniformly to create a new dataset. We refer to this dataset as the coarse dataset X_c and the original dataset as the dense dataset X for the rest of the paper. Various techniques for data reduction have been proposed, such as dimensionality reduction and numerosity reduction. Considering to transfer the partial network model pre-trained with the coarse dataset to continue to train with the dense model, we keep the same



(a) CosmoFlow (b) DeepCAM

Fig. 1. Schematic of convolutional neural network used in CosmoFlow and DeepCAM. The layer that is different between the dense and the coarse model is marked in yellow. The input sample sizes and the output sizes of the last convolutional layer of the dense and the coarse model are shown on the left and right columns, respectively.

number of dimensions of the data samples and adopt a simple strategy to reduce the data resolution. For each data dimension, we keep the number of channels unchanged if they represent different figures and reduce the size of other dimensions based on their scientific features. For example, a training sample x_i from the CosmoFlow dataset is a 4D array, where the first three dimensions denote the 3D matter distribution, and the last one consists of 4 channels representing 4 different redshifts. In this example, we reduce the data resolution along the first three dimensions of each sample by averaging every 2^3 neighbors of a data point. The resulting coarse sample is referred to as x_{ci}. As for the DeepCAM dataset, climate variables are stored on 1152 x 768 spatial grids in the data sample x_i. There are 16 pixel-wise variables for each spatial grid containing wind speed, temperature, pressure, precipitation, etc. Thus, for each climate variable, we calculate the average values of each adjacent 2 x 2 spatial grid to create a coarse sample x_{ci}. Note that the size of labels can be related or unrelated to the size of data samples. In this case, we keep the same labels Y for the coarse dataset as the dense dataset.

B. Model Architecture for Multi-Resolution Training

We adjust the original model in order to train with coarse data. In the rest of the paper, we refer to the original model to be trained with the original resolution data as the dense model, $\mathcal{M}$, and the adjusted model to be trained with reduced-resolution data as coarse mode, $\mathcal{M}_c$. With the smaller input size, the output size of the last convolutional layer also becomes smaller. Recent work on transfer learning suggests parameters in the neural network usually converge from the

input side to the output side layers [30]. Thus, when constructing the coarse model $\mathcal{M}_c$ based on the dense model $\mathcal{M}$, we keep the structures of the input side layers the same and only adjust the output side layers. As typical convolutional neural networks are composed of alternate convolution layers and pooling layers with or without several fully connected layers at the end [31], we accordingly categorize them into two types: with or without fully connected layers. For models consisting of convolution layers at the input side and fully connected layers at the output side, we only adjust the size of the weights of the first fully connected layer without adding extra layers. For example, for the model architecture of the CosmoFlow in Figure 1 (a), as shown on the right side, using the coarse sample x_{ci} causes the output size of the last convolutional layer to be smaller than the one in the dense model, which is shown on the left side. Thus, we keep the size of weights in the convolutional layers and the output side fully connected layers marked in orange unchanged and only adjust the size of weights of the fully connected layer marked in yellow for the coarse model $\mathcal{M}_c$.

For fully convolutional networks, the output size of the last layer reduces as the input sample size becomes smaller. Thus, we add an extra upsampling layer at the output side to adjust outputs to match the size of the labels Y. For example, DeepCAM implements a convolutional encoder-decoder segmentation model, which is shown in [19]. From the simplified structure shown in figure 1 (b), we can see that with a coarse sample as the input, the size of the output of the decoder is proportionally reduced. Thus, we add an extra upsampling layer at the output side in the coarse model to fill the new grids with values from the nearest neighbors. Without making changes to the labels, we use the same loss function for both the coarse and the dense model.

C. Switching Mechanism

Using the transfer learning techniques, we pass the weights of convolution layers from the pre-trained coarse model $\mathcal{M}_c$ to initialize the corresponding layers in the dense model $\mathcal{M}$. The weights of the rest of the layers are randomly initialized. For example, in the CosmoFlow case, the weights of all convolution layers are transferred from the coarse model to the dense model, while the weights of all fully connected layers are randomly initialized. In our implementation, after training the coarse model on GPUs, we first load the coarse model onto the CPUs. Then, the transferred weights are copied from the coarse model to the dense model. Finally, the dense model is offloaded onto the GPUs to be further trained with the dense dataset.

In transfer learning, people sometimes fine-tune all parameters in the model or freeze some top (output side) layers. In our multi-resolution training, because the pre-trained coarse model was trained using data X_c containing less information compared to the original set X, freezing layers may affect the generalization performance. Thus, we fine-tune all the weights in the dense model $\mathcal{M}$ using the dense set X without freezing any layers.

In multi-resolution training, finding a proper switching point is important. We need to decide how many epochs to train on the coarse model $\mathcal{M}_c$ before switching to the dense model $\mathcal{M}$. Without sufficient training, transferred weights may have little effect on the dense model. However, training in the first stage for too long can add much extra cost to the overall training time. Thus, we propose a switching mechanism to decide when to switch from training on the coarse model to the dense one. Specifically, we look for the turning point of the training loss curve by measuring the reduction of minimum loss in consecutive epochs. When the reduction in the recent $\mathcal{T}$ epochs is smaller than the current threshold ϵ, we stop the training on the coarse set X_c and switch the model. The threshold ϵ is set based on the peak magnitude of the loss function, which can be tuned to adapt to different tasks and datasets.

D. Parallelization

Under settings described earlier, we adopt synchronous SGD with data parallelism in our training. For each iteration, one mini-batch is evenly assigned to all processes to compute the gradients locally. Once the data has been processed, all the workers use inter-process communication to average the gradients, done by all-reduce communications. Then, all the workers use the synchronized gradients to update their local models. Thus, the communication cost is proportional to the size of the model and the number of GPUs.

It is common to shuffle the training samples per epoch. We restrict the inter-process shuffling to reduce expensive I/O costs. Given N training samples, B as the global batch size, and P as the number of processes, we evenly divide samples into P groups and randomly assign each process with one group of samples per epoch. Then, each individual process randomly reads $\frac{B}{P}$ samples from the assigned samples at each iteration. In Section IV, we will further analyze the parallelization performance.

IV. Evaluation

In this section, we evaluate the performance of our multi-resolution training strategy using two real-world scientific applications: CosmoFlow and DeepCAM. All the experiments are conducted on two supercomputers: Summit at ORNL and Cori at NERSC, with different hardware configurations.

A. Experimental Setup

We conduct experiments on two large-scale HPC platforms, Cori and Summit. Cori is a Cray XC40 supercomputer that has 18 nodes for GPU machines. Each node has two sockets of Intel Xeon Gold 6148 (Skylake) CPUs, 8 NVIDIA V100 GPUs, and 384 GB memory space. Summit is an IBM AC922 system that consists of 4,608 nodes. Each node has two sockets of IBM Power9 CPUs, 6 NVIDIA V100 GPUs, and 512 GB memory space.

For CosmoFlow, we use IBM Watson Machine Learning Community Edition 1.7.0-3, which supports TensorFlow 2.1.0 and Horovod 0.19.0 on Summit. On Cori, we use TensorFlow

2.2.0 and Horovod 0.19.0. For DeepCAM, we follow the source code from MLPerf HPC reference implementations at [32] using PyTorch. We use PyTorch 1.7.1 and DistributedDataParallel (DDP) from Apex on Summit and Cori. Our experiments use 32 to 128 GPUs and have one MPI rank per GPU allocated (i.e., 6 ranks per node on Summit and 8 ranks per node on Cori).

On Cori, the Lustre parallel file system is used to store the datasets. On Summit, our datasets are stored on Alpine, which is a POSIX-based IBM Spectrum Scale parallel file system.

Model architectures – We use the adapted neural network models based on the ones provided in the MLPerf HPC benchmark suite. For CosmoFlow, we use a modified version of Livermore Big Artificial Neural Network (LBANN), consisting of 7 3-D convolutional layers followed by 3 fully connected layers. For DeepCAM, we use the modified DeepLabv3+ network, which consists of an Xception network [33] as an encoder, atrous spatial pyramid pooling (ASPP) [34] blocks, and a decoder.

When switching from the coarse model to the dense model, based on III-C, we transferred the learned weights of convolution layers from the coarse model. For CosmoFlow, the weights of fully connected layers are randomly initialized. Then, the dense models are trained on the dense dataset by fine-tuning all trainable parameters.

B. Performance Results of CosmoFlow

CosmoFlow is a deep learning tool for Cosmology data analysis. The training data of CosmoFlow are simulated 3-dimensional distributions of masses with different initial conditions. For each initial condition, there are 4 channels representing the evolved universe with 4 red-shift values. Each sample represents the simulated problem domain, which represents the universe by binned into a cube of size 512 x 512 x 512. Then, the cubes are further reshaped into 128 x 128 x 128 x 4 by concatenating the binned cubes from 4 red-shifts on channel dimension. Given the mass distributions, CosmoFlow estimates 4 initial conditions of the universe. Thus, each sample size is 128 x 128 x 128 x 4, and the label size is 4. There are 80 HDF5 files and each file contains 128 samples, which are split into 80% training, 10% validation, and 10% test sets. These files are generated from the same source files as CosmoFlow from the MLPerf HPC training benchmark suite.

In the MLPerf HPC benchmarks, the CosmoFlow model is trained with the standard SGD optimizer. The loss function is Mean Square Error (MSE), and the initial learning rate is 0.001, which is dropped to 2.5×10^{-4} and 1.25×10^{-4} at 32 and 64 epochs. The global batch size is set to 64. The target quality used in the MLPerf HPC benchmark is mean-absolute-error (MAE) < 0.124.

We adjusted the settings and trained the model using Adam optimizer [13]. We set the global batch size to 256 and used 0.002 as the initial learning rate. The learning rate is decayed twice with a factor of 0.1 at 50 and 75 epochs. We evaluate our multi-resolution training method over training with the original dataset with CosmoFlow on Summit using 32 and 64 GPUs. Given data samples with the size 128 x 128 x 128 x 4, we divide the cubes of size 128 x 128 x 128 into small cubes of size 2 x 2 x 2 and replace the small cubes with the sum of 8 values in them to get the coarse samples of size 64 x 64 x 64 x 4. The best-tuned model using the original dataset could achieve the validation loss (MSE) of 0.0025 and validation loss (MAE) of 0.023. Thus, we set 0.0025 as the target validation loss (MSE) to decide when to stop the training.

TABLE I
THE VALIDATION LOSS, TRAINING EPOCH, TOTAL TRAINING TIME (ON CORI AND SUMMIT) FOR COSMOFLOW WITH 32 GPUS. THE TIMINGS ARE ALL IN SECONDS. MRT REDUCES THE TRAINING TIME BY 30% OVER THE BASELINE (DENSE).

Dataset	Validation loss (MSE)	Number of epochs	Total time (Cori)	Total time (Summit)
Coarse	0.0066	85	145.35	148.75
Dense	0.0025	88	1073.60	858.00
MRT	0.0025	55 (coarse) + 54 (dense)	752.85	622.75

TABLE II
THE VALIDATION LOSS, TRAINING EPOCH, TOTAL TRAINING TIME (ON CORI AND SUMMIT) FOR COSMOFLOW WITH 64 GPUS. THE TIMINGS ARE ALL IN SECONDS. MRT REDUCES THE TRAINING TIME BY 27% OVER THE BASELINE (DENSE).

Dataset	Validation loss (MSE)	Number of epochs	Total time (Cori)	Total time (Summit)
Coarse	0.0066	85	102.00	90.95
Dense	0.0025	88	561.44	483.12
MRT	0.0025	55 (coarse) + 54 (dense)	410.52	355.31

We refer to **baseline** case as the original neural network models trained with only the dataset in the original resolution, i.e., dense data. Tables I and II show the performance of the baseline, training only with the coarse dataset, and our proposed multi-resolution training strategy (MRT) in terms of validation loss, the number of training epochs, and the total training time on both Cori and Summit. We average the results over five random seeds for each training strategy. First, as we expected, training only with the coarse dataset takes a much shorter time than training with the original dataset. However, with the best-tuned hyper-parameters, the end validation loss is 0.0064, which cannot achieve a similar one as the baseline. Because the coarse dataset is generated by summing up the adjacent elements, it does not contain enough information for more accurate predictions. Second, we can see that the proposed multi-resolution training strategy reduces the training time by 29.87% on Cori and 27.42% on Summit. This end-to-end training time comparison clearly shows how effectively our proposed training strategy speeds up the neural network training.

Figure 2 shows the training and validation loss curves of the baseline and our proposed MRT strategy. From figure

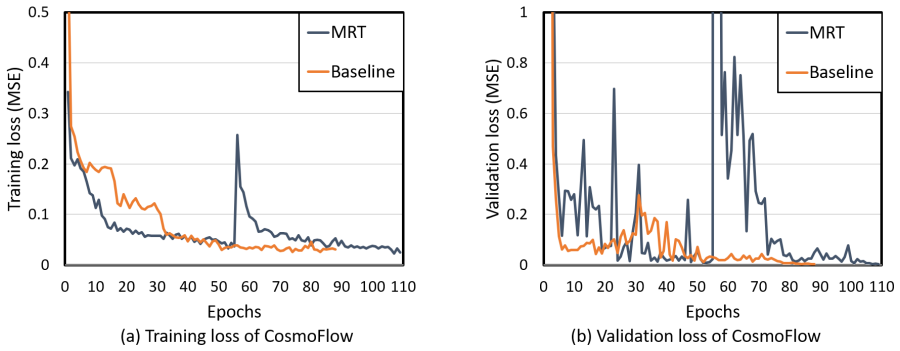


Fig. 2. The learning curves of CosmoFlow of the baseline and our proposed MRT strategy. The global batch size is 256 and the learning rate is 0.002. We used Adam optimizer. Using Mean Squared Error (MSE) metric, the achieved validation loss is 0.0025.

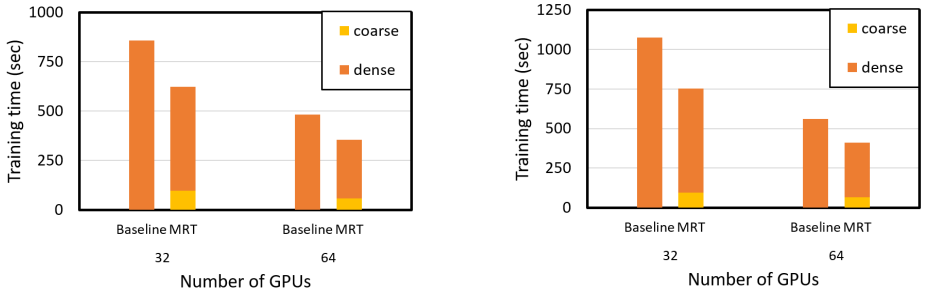


Fig. 3. Comparison of the training timing breakdown for CosmoFlow between baseline and the proposed MRT strategy on Summit (a) and Cori (b).

2, first, we can see that the loss curves of MRT have a spike when switching the model. Because the fully connected layers are newly initiated, and the input data is different, the model takes a few epochs to adjust to the data samples with different resolutions. Second, comparing the curves of the second training stage of MRT with the first half curves of the baseline, after training for about 20 epochs, both losses of our proposed strategy become smaller than the corresponding values in the baseline. This shows that transferred knowledge from the coarse model boosts the training with the dense dataset after switching the model without affecting the end validation accuracy.

TABLE III
THE AVERAGE EPOCH TIMING BREAKDOWN FOR COSMOFLOW ON SUMMIT. THE TIMINGS ARE ALL IN SECONDS.

Number of GPUs	Dataset	I/O time	Comm time	Comp time	Average epoch time
32	**Coarse**	0.04	0.23	1.48	1.75
	Dense	0.00	0.44	9.31	9.75
64	**Coarse**	0.00	0.28	1.07	1.35
	Dense	0.04	0.59	4.86	5.49

Scaling performance – The two supercomputers, Summit and Cori, have different hardware configurations. With different settings of GPUs, we expect different computation times per GPU. Also, different communication networks and file system settings can affect the communication time and the I/O time. Thus, we measure and present the performance of CosmoFlow on both supercomputers. We use the same hyper-parameter settings and train both models with three random seeds to calculate the mean values.

1) Summit GPU Nodes: Figure 3 compares the end-to-end training time for CosmoFlow between the baseline and our strategy using 32 and 64 processes on Summit. The model does not fit into fewer than 32 GPUs. Thus, we present the scaling performance from 32 processes (GPUs). The training time on the coarse model is marked in yellow. The figure shows that training on the coarse model only takes a small portion of time. When using 64 GPUs on Cori, the learned knowledge helps largely reduce the training time on the dense model (344.52 sec) to achieve the target validation loss compared to the baseline (561.44 sec). Therefore, the total training time of our proposed training strategy ends up being shorter than the baseline.

Table III presents the scaling performance of CosmoFlow on Summit GPU nodes. First of all, as data samples are prefetched for each epoch, most of the I/O time for training with the coarse or the dense dataset is overlapped with the computation time. For the dense model, when increasing the number of GPUs to 64, partial I/O cost is exposed due to reduced computation time. Second, though the input data size becomes 1/8 for the coarse model, the sizes of the weights of convolutional layers and the last two fully connected layers are the same. The weight size of the first fully connected layer becomes 1/8 for the coarse model. Thus, the average

communication cost per epoch of the coarse model is around half of training with the dense set. When using 64 GPUs, the communication cost increases as more processes are involved in the synchronization at each iteration. Third, we can see that the computation time significantly reduces when training on the coarse dataset. Because of the small input data size, training with the coarse dataset has much lower computation cost.

TABLE IV
THE AVERAGE EPOCH TIMING BREAKDOWN FOR COSMOFLOW ON CORI. THE TIMINGS ARE ALL IN SECONDS.

Number of GPUs	Dataset	I/O time	Comm time	Comp time	Average epoch time
32	**Coarse**	0.09	0.13	1.49	1.71
	Dense	1.27	0.21	10.72	12.20
64	**Coarse**	0.05	0.19	0.96	1.20
	Dense	0.37	0.19	5.67	6.38

2) Cori GPU Nodes: We perform the same CosmoFlow experiments on Cori. Table IV and figure 3 (b) show the performance results of CosmoFlow on Cori. The timing breakdown shows similar performance results compared to the results on Summit. Note that due to different hardware configurations, the computation time of the dense model on Cori is longer than that on Summit, while it is the opposite for the coarse model results. Thus, the training time of the coarse model takes a smaller portion of the total time than that on Summit.

C. Performance Results of DeepCAM

DeepCAM is a deep learning tool for the segmentation of extreme weather phenomena. The model is trained with the CAM5 dataset, which contains simulated climate variables stored on an 1152 x 768 spatial grid. For each grid, the sample contains 16 channels representing water vapor, precipitation, pressure, etc. The grid-level mask labels are generated with the Toolkit for Extreme Climate Analysis and a flood fill algorithm. They correspond to 3 classes: Tropical Cyclone (TC), Atmospheric River (AR), and background (BG) class. So, each sample size is 1152 x 768 x 16, and the corresponding label size is 1152 x 768. The CAM5 dataset has 63K samples in total, which are split into 80% training, 10% validation, and 10% test samples.

Following the same settings used in the MLPerf HPC Training v0.7 benchmark [21], we trained a modified version of DeepLabv3+ network on the CAM5 dataset using LAMB optimizer [35], the layer-wise adaptive optimizer for large-batch training. For the segmentation accuracy, we use the intersection over union (IoU) metric, which measures how much the given two regions are overlapped with each other. In the original publication describing this application [19], the IoU accuracy achieved is 73%. In the MLPerf HPC Training v0.7 benchmark [21], DeepCAM is trained until reaching the quality target, 0.82 of the validation IoU between the predictions and the targets. We adopt the same target validation accuracy to train the model until the validation accuracy reaches 0.82. Because of the class imbalance, DeepCAM uses the weighted cross-entropy loss. Due to the GPU memory limitation, we use 2 as the local batch size for the dense model. For the baseline and our proposed MRT strategy, we present results with the best-tuned hyper-parameter settings. We use 0.001 as the initial learning rate and decay the learning rate by a factor of 10. We evaluate the performance on Summit and Cori using 64 and 128 GPUs. With the size of the original set as 1152 x 768 x 16, we preprocess the samples to the coarse dataset with the size 576 x 384 x 16.

TABLE V
THE VALIDATION ACCURACY, TRAINING STEP, THE TOTAL TIME ON CORI, AND THE TOTAL TIME ON SUMMIT FOR DEEPCAM WITH 64 GPUS. THE TIMINGS ARE ALL IN SECONDS. MRT REDUCES THE TRAINING TIME BY 23% OVER THE BASELINE (DENSE).

Dataset	Validation accuracy	Number of iterations	Total time (Cori)	Total time (Summit)
Coarse	0.75	4500	2301.29	2318.26
Dense	0.82	8900	4924.79	4324.53
MRT	0.82	1888 (coarse) + 5100 (dense)	3787.59	3450.74

TABLE VI
THE VALIDATION ACCURACY, TRAINING STEP, THE TOTAL TIME ON CORI, AND THE TOTAL TIME ON SUMMIT FOR DEEPCAM WITH 128 GPUS. THE TIMINGS ARE ALL IN SECONDS. MRT REDUCES THE TRAINING TIME BY 18% OVER THE BASELINE (DENSE).

Dataset	Validation accuracy	Number of iterations	Total time (Cori)	Total time (Summit)
Coarse	0.75	4500	1326.16	1552.50
Dense	0.82	4500	2507.44	2159.24
MRT	0.82	1652 (coarse) + 2800 (dense)	2047.04	1913.47

The computational efficiency is affected if we keep using a small local batch size when training with the coarse dataset. Thus, very limited time reduction is gained compared to using the dense dataset. Reduced sample sizes allow a larger number of samples assigned to one worker, so we increase the local batch size to 8 when training with the coarse set. We use 250 as the patience and 0.001 as the improvement threshold on training loss to decide when to switch the model. When the improvement in training loss is smaller than the threshold for a number of iterations as the patience, we switch the model at the end of the current training epoch.

Table V and Table VI present the validation accuracy, the number of training iterations, the training time comparison among the baseline, the training only with the coarse dataset, and our proposed MRT strategy. First, since less information is included in the coarse samples for a specific geographic region, the best validation accuracy that can be achieved with the coarse dataset is reduced to 0.75. Second, we can see that our training strategy achieves the same accuracy with 23.09% reduced end-to-end training time on Cori and 20.20% reduced time on Summit compared to the baseline. It indicates that our

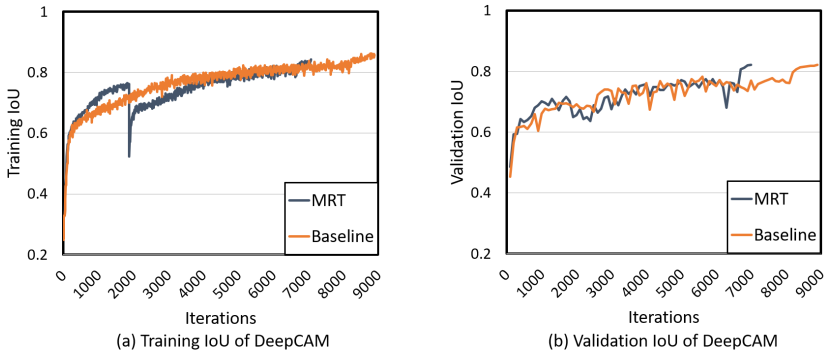


Fig. 4. The training and validation accuracy (IoU) curves of DeepCAM of the baseline and our proposed MRT strategy. The global batch size is 128 and the learning rate is 0.001. We used LAMB optimizer. Using the Intersection over Union (IoU) metric, the achieved validation accuracy is 0.82.

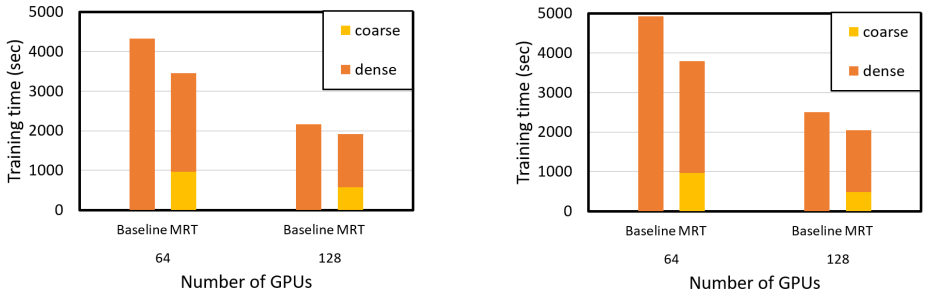


Fig. 5. Comparison of the training timing breakdown for DeepCAM between baseline and the proposed MRT strategy on Summit (a) and Cori (b).

method not only boosts the training efficiency of regression problems like CosmoFlow but also improves the performance of pixel-level segmentation problems.

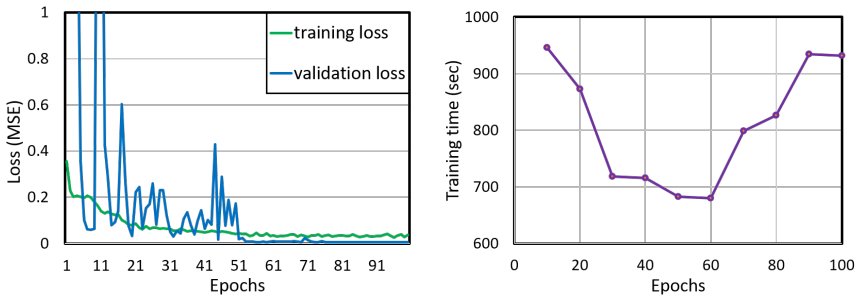
Figure 4 shows the training and validation accuracy calculated with IoU of the baseline and our proposed training strategy. The training accuracy is measured for every 10 iterations (steps), and the validation accuracy is for every 100 iterations. Similar to the curves of CosmoFlow, there is a spike when switching between the coarse and the dense models. Comparing the baseline with the second phase of training with the MRT strategy, we can see that both the training and validation accuracy curves of our proposed MRT strategy are higher than the baseline. Also, with the proposed strategy, the same accuracy is achieved with fewer training iterations. These results demonstrate that our proposed training strategy reduces the end-to-end training time without losing accuracy.

TABLE VII
THE AVERAGE EPOCH TIMING BREAKDOWN FOR DEEPCAM ON SUMMIT. THE TIMINGS ARE ALL IN SECONDS.

Number of GPUs	Dataset	I/O time	Comm time	Comp time	Average epoch time
64	**Coarse**	3.15	8.24	110.19	121.58
	Dense	8.23	30.69	421.23	460.15
128	**Coarse**	2.29	10.38	68.75	81.42
	Dense	4.89	16.19	205.88	226.96

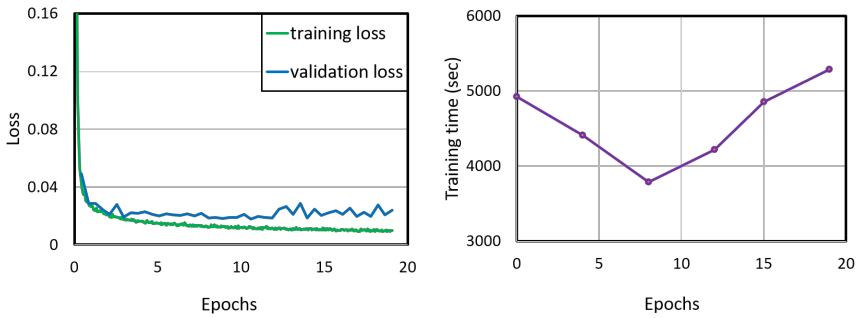
Scaling performance – We also study the scaling performance of the DeepCAM application on Summit and Cori. Compared to CosmoFlow, DeepCAM has a significantly larger model. Thus, we expect a different performance impact of our proposed training strategy on DeepCAM. Note that the model does not fit the memory space when using fewer than 64 GPUs; thus, we scale up the training to 128 GPUs.

1) Summit GPU nodes: Figure 5 presents the scaling performance of DeepCAM on Summit. We can see how our proposed method affects the end-to-end training time. Training with the coarse dataset has a much cheaper average epoch time than training with the dense dataset. Though training with the coarse dataset brings extra cost, the wall-clock time is reduced due to the reduced training steps on the dense dataset to achieve the same validation accuracy. Table VII shows the timing breakdown of the average training time per epoch for both the coarse and the dense datasets. We can see that the computation time and exposed I/O time are significantly reduced for the coarse model. Because in the coarse model, the input samples are 1/4 size of the dense samples, which reduces the computation cost. The communication time of the coarse model is also reduced compared to the time of the dense model, which corresponds to what we expected. Since the local batch size increases when using the coarse dataset, the number of iterations per epoch is reduced accordingly. Considering that the structure of layers with trainable weights is not changed between two models, the amounts of gradients averaged per iteration are the same. However, due to fewer synchronizations conducted for the coarse model, the communication time



(a) (b)

Fig. 6. (a) The learning curves of CosmoFlow on the coarse model. (b) The end-to-end training times for different epoch numbers when switching the model.



(a) (b)

Fig. 7. (a) The learning curves of DeepCAM on the coarse model. (b) The end-to-end training times for different iteration numbers when switching the model.

becomes shorter.

TABLE VIII
THE AVERAGE EPOCH TIMING BREAKDOWN FOR DEEPCAM ON CORI. THE TIMINGS ARE ALL IN SECONDS.

Number of GPUs	Dataset	I/O time	Comm time	Comp time	Average epoch time
64	**Coarse**	2.19	3.66	114.84	120.69
	Dense	8.96	19.14	495.92	524.02
128	**Coarse**	2.39	8.92	58.25	69.55
	Dense	4.09	12.78	246.69	263.56

2) Cori GPU nodes: We compare the performance of DeepCAM adopting the same settings on Cori. Figure 5 (b) presents the scaling performance of DeepCAM on Cori. The end-to-end training time shows similar performance results to that on Summit GPU nodes. Table VIII presents the timing breakdown of training with the coarse and the dense model on Cori. Similar analysis has been given with the observation of the same trend of the largely reduced time on the coarse model.

D. Impact of Switching Point

Picking a proper epoch number to switch from the coarse to dense model is critical to the end-to-end training time. In order to study the impact of the model switch points, we repeated the entire training 10 times, each using a different switching epoch number. This experiment is used to compare against the automatic switching approach that uses a threshold and patience on the training loss. In our case, we set the patience to 5 epochs and the improvement threshold to 0.001.

Figure 6 (b) shows the end-to-end training times of CosmoFlow when switching from coarse to dense models at a given epoch number. The curve points out that the best end-to-end training time (the lowest value) is achieved when switching at epoch 60, corresponding to the switching point calculated based on the patience and threshold. From the loss curves shown in Figure 6 (a), we can see both the training and validation losses become relatively flat at around epoch 60. In addition, we observe that when the switching epoch number is smaller than 30, the cost of dense model training is much higher than others. This is because the coarse model does not receive enough training, resulting in the learned features with little effect on the training for the dense model. When the switching epoch number is larger than 80, the end-to-end time also dramatically increases. This is because the training and validation losses do not decrease after epoch 80. Continuing the training of the coarse model beyond that prolongs the time to convergence for the dense model. Figure 7 shows the loss curves of training on the coarse model and end-to-end training times of DeepCAM when switching at different numbers of iterations. We can see similar patterns of the training times at different switching points. These results demonstrate that measuring the loss improvement in consecutive epochs and switching the model when the improvement is smaller than a preset threshold for a number of epochs as the patience can lead to the switching point with a relatively shorter end-to-end

training time.

V. Conclusion

In the paper, we discussed that given a scientific dataset that can be represented as different resolutions, how to take advantage of this feature to reduce neural network training time. We proposed a multi-resolution training strategy that transfers knowledge from the coarse dataset to accelerate the training on the original problem. We applied our proposed strategy to two real-world scientific applications. Our experimental results demonstrate that the proposed multi-resolution training can reduce the end-to-end training time while maintaining the model accuracy. Considering data of different resolutions can be generated from the original data, further exploring the interactions between data of different resolutions and utilizing more than two resolutions can be interesting future work.

Acknowledgment

This work is supported in part by U.S. Department of Energy under award numbers DE-SC0021399, DE-SC0019358, and the National Institute of Standards and Technology award number 70NANB19H005. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC awards ASCR-ERCAP0021094 and ASCR-ERCAP0021411. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] A. Piccione, J. Berkery, S. Sabbagh, and Y. Andreopoulos, "Physics-guided machine learning approaches to predict the ideal stability properties of fusion plasmas," *Nuclear Fusion*, vol. 60, no. 4, p. 046033, mar 2020. [Online]. Available: https://doi.org/10.1088/1741-4326/ab7597

[2] A. Agrawal and A. Choudhary, "Deep materials informatics: Applications of deep learning in materials science," *MRS Communications*, vol. 9, no. 3, pp. 779–792, 2019.

[3] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook *et al.*, "Cosmoflow: Using deep learning to learn the universe at scale," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 819–829.

[4] D. George and E. Huerta, "Deep neural networks to enable real-time multimessenger astrophysics," *Physical Review D*, vol. 97, no. 4, p. 044039, 2018.

[5] T. Kurth, J. Zhang, N. Satish, E. Racah, I. Mitliagkas, M. M. A. Patwary, T. Malas, N. Sundaram, W. Bhimji, M. Smorkalov *et al.*, "Deep learning at 15pf: supervised and semi-supervised classification for scientific data," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–11.

[6] Y. You, J. Hseu, C. Ying, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large-batch training for lstm and beyond," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–16.

[7] T. Akiba, S. Suzuki, and K. Fukuda, "Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes," *arXiv preprint arXiv:1711.04325*, 2017.

[8] N. Dryden, N. Maruyama, T. Moon, T. Benson, M. Snir, and B. Van Essen, "Channel and filter parallelism for large-scale cnn training," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–20.

[9] Y. Ma, F. Rusu, K. Wu, and A. Sim, "Adaptive elastic training for sparse deep learning on heterogeneous multi-gpu servers," 2021.

[10] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in Neural Information Processing Systems*, vol. 30, pp. 1709–1720, 2017.

[11] M. Chen, Z. Yan, J. Ren, and W. Wu, "Standard deviation based adaptive gradient compression for distributed deep learning," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 529–538.

[12] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

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

[14] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[15] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," *Advances in neural information processing systems*, vol. 26, pp. 315–323, 2013.

[16] L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takáč, "Sarah: A novel method for machine learning problems using stochastic recursive gradient," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2613–2621.

[17] S. R. Fulton, P. E. Ciesielski, and W. H. Schubert, "Multigrid methods for elliptic problems: A review," *Monthly Weather Review*, vol. 114, no. 5, pp. 943–959, 1986.

[18] I. Suisalu and E. Saar, "An adaptive multigrid solver for high-resolution cosmological simulations," *Monthly Notices of the Royal Astronomical Society*, vol. 274, no. 1, pp. 287–299, 1995.

[19] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica *et al.*, "Exascale deep learning for climate analytics," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 649–660.

[20] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang *et al.*, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[21] S. Farrell, M. Emani, J. Balma, L. Drescher, A. Drozd, A. Fink, G. Fox, D. Kanter, T. Kurth, P. Mattson *et al.*, "Mlperftm hpc: A holistic benchmark suite for scientific machine learning on hpc systems," *arXiv preprint arXiv:2110.11466*, 2021.

[22] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*. Springer, 2018, pp. 270–279.

[23] D. Jha, K. Choudhary, F. Tavazza, W.-k. Liao, A. Choudhary, C. Campbell, and A. Agrawal, "Enhancing materials property prediction by leveraging computational and experimental data using deep transfer learning," *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

[24] V. Gupta, K. Choudhary, F. Tavazza, C. Campbell, W.-k. Liao, A. Choudhary, and A. Agrawal, "Cross-property deep transfer learning framework for enhanced predictive analytics on small materials data," *Nature communications*, vol. 12, no. 1, pp. 1–10, 2021.

[25] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*. PMLR, 2014, pp. 647–655.

[26] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2661–2671.

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

[28] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "Spottune: transfer learning through adaptive fine-tuning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4805–4814.

[29] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.
[30] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein, "Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability," *arXiv preprint arXiv:1706.05806*, 2017.
[31] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.
[32] "Mlperf hpc benchmark suite," 2021. [Online]. Available: https://github.com/mlcommons/hpc
[33] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.
[34] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.
[35] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," *arXiv preprint arXiv:1904.00962*, 2019.