Dissecting Convolutional Neural Networks for Runtime and Scalability Prediction

Tim Beringer
tim.beringer@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Hesse, Germany

Jakob Stock
jakob.stock@stud.tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Hesse, Germany

Arya Mazaheri
arya.mazaheri@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Hesse, Germany

Felix Wolf
felix.wolf@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Hesse, Germany

ABSTRACT

Given the computational complexity of deep neural networks (DNN), accurate prediction of their training and inference time using performance modeling is crucial for efficient infrastructure planning and DNN development. However, existing methods often predict only the inference time and rely on exhaustive benchmarking and fine tuning, making them time consuming and restricted in scope. As a remedy, we propose ConvMeter, a novel yet simple performance model that considers the inherent characteristics of DNNs, such as architecture, dataset, and target hardware, which strongly affect their runtime and scalability. Our performance model, which has been thoroughly tested on convolutional neural networks (ConvNets), a class of DNNs widely used for image analysis, offers the prediction of inference and training time, the latter on one or more compute nodes. Experiments with various ConvNets demonstrate that our runtime predictions of inference and training phases achieved an average error rate of less than 20% and 18%, respectively, making the assessment of ConvNets regarding efficiency and scalability straightforward.

CCS CONCEPTS

• **Computer systems organization → Neural networks**; • **Computing methodologies → Modeling and simulation**.

KEYWORDS

Artificial intelligence, deep neural networks, convolution, performance modeling, distributed training, scalability

ACM Reference Format:
Tim Beringer, Jakob Stock, Arya Mazaheri, and Felix Wolf. 2024. Dissecting Convolutional Neural Networks for Runtime and Scalability Prediction. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3673038.3673107

1 INTRODUCTION

Deep neural networks (DNN) and, in particular, convolutional neural networks (ConvNet) are the mainstream machine-learning methods for a wide variety of computer-vision tasks, such as image classification, object detection, and semantic segmentation. The reason for their widespread use is their ability to achieve high accuracy on various datasets, while requiring reasonable computational resources. Moreover, their increasing success, even on edge devices with relatively low computational resources, is largely derived from a faster training infrastructure (e.g., DNN-specific training scheduler [15, 18]) and iterative DNN architecture improvement, either manually [7] or automatically using neural architecture search (NAS) [5] and design space exploration [19, 20]. NAS can significantly reduce the time and effort required to design hardware-aware DNNs, yet requires extensive computational capacity due to larger model sizes [2, 3, 28] or many rounds of trial trainings [16]. Additionally, DNN optimization methods such as network pruning and quantization often require extra fine-tuning steps to recover some of the lost performance by adjusting the remaining weights.

The effective operation of DNN training schedulers and NAS, as well as most network optimization techniques, commonly depend on or can profit from a performance prediction tool capable of precisely estimating the non-functional properties, such as training or inference time and scaling factor. Performance modeling is a well-known technique that involves developing mathematical models to predict the performance of a workload under different conditions. These models can help determine the appropriate hardware and software resources required for a system to meet its performance goals. Furthermore, recent DNN models and datasets have rapidly increased in size, making low-cost prior estimation of runtime essential for such methods and even for machine-learning developers to choose the appropriate hardware setup for running or training their DNNs. Particularly, an accurate performance model can assist in reducing the training cost by choosing the training parameters (e.g., batch size, number of computing devices) and the computing infrastructure, such as cloud instances.

Existing DNN-specific performance models are limited in their capability, as they mainly focus on predicting only the inference time of the overall DNN without a detailed analysis of the constituent blocks—a crucial feature needed by NAS methods. Moreover, many of the existing methods are based on empirical performance modeling [10, 30, 32], requiring numerous data points collected from the target device to train their model. Therefore, these methods incur a significant modeling cost to the user, an investment that might need to be repeated for each new hardware or DNN configuration. Such drawbacks triggered the following research questions:

- Can we obtain a holistic DNN performance model capable of predicting both training and inference time?
- Can we leverage the inherent characteristics of a DNN to simplify performance modeling?
- Can we predict the DNN training scalability in a distributed environment using a performance model?

In this paper, we answer the above questions for convolutional neural networks (ConvNets), a class of DNNs widely used for image analysis, by introducing ConvMeter, a novel yet simple performance model. Our performance model predicts the training and inference time of ConvNets in single-node and distributed training schemes. In contrast to existing methods, we use ConvNet metrics, such as inputs, activations, and FLOPs, that can be easily computed without running the ConvNet. We demonstrate that a simple performance model, relying neither on advanced techniques like machine learning nor extensive blind benchmarking, can predict ConvNet runtimes with reasonably high accuracy. Furthermore, our method does not require multiple iterations of benchmarks to fine-tune the prediction model. The structure of the performance model adapts well to the desired target hardware, as the tunable coefficients capture the overall runtime performance differences between different hardware platforms, and the previously mentioned ConvNet metrics adapt to the differences between the memory bandwidth and computational performance of the device. Our experiments validate that our method achieves notable accuracy for predicting both inference and training time on different hardware (i.e., CPUs and GPUs) and node setups (i.e., single node or multiple nodes). Moreover, the performance analysis can be used for analyzing the scalability of ConvNets. In essence, this paper makes the following major contributions:

- The identification of inherent ConvNet metrics (FLOPs, Inputs, and Outputs) as the foundation of efficient yet effective performance prediction without extensive benchmarking.
- A simple yet accurate performance model for ConvNets that enables the runtime prediction of DNN inference and training on CPUs or GPUs.
- Scalability prediction of distributed training as a function of the number of computing devices and the batch size.

2 BACKGROUND

Modern ConvNets consist of recurring blocks with multiple layers (convolutional, pooling, fully connected). Convolution operations are usually followed by batch normalization and activation functions. As networks increase in depth, they require more parameters, leading to longer training and inference times. Furthermore, convolutional layers are the most time-consuming layers during computation, as they apply a filter on an image tensor with size $B \times C_{in} \times H_{in} \times W_{in}$ and produce an output with the size of $B \times C_{out} \times H_{out} \times W_{out}$, where B is the batch size, $C_{\{in,out\}}$ are input/output channels, and $\{H, W\}_{\{in,out\}}$ are input/output image size.

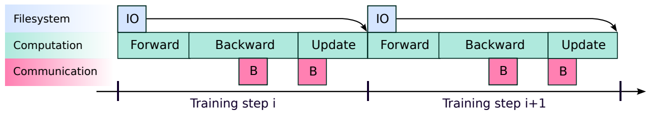

Figure 1: Visualization of a synchronous training step and its phases [13]. IO: Reading the (mini-)batch for the next training step. B: Bucket, consisting of gradient updates.

DNN training consists of iteratively adjusting a deep network's parameters for a given dataset until the obtained model achieves the desired accuracy. Although the training can be performed using a single processor (e.g. GPU), it is often required to scale the training to multiple processors/nodes to accommodate large DNNs or datasets that would take many days to be processed. Distributed DNN training employs different types of parallelism to train large-scale models effectively. However, within the scope of this paper, we mainly focus on data parallelism with weak scaling as one of the preferred methods to expand the training of ConvNets to multiple GPUs. Within this method, the parameters are synchronized with the other devices, using various techniques such as parameter server or all-reduce strategy. All-reduce strategy is more widely used in distributed training due to its faster convergence, scalability, low communication overhead, and flexibility. Horovod [24], for instance, is a distributed deep learning framework that supports an all-reduce strategy for large-scale model training.

During the training, the dataset is processed multiple times, where each pass is called an epoch. In data parallelism, the dataset is split into equally-sized portions called batches within an epoch. As described by Pauloski et al. [13] and shown in Figure 1, each training step in processing a batch consists of three phases. During the forward pass, a DNN calculates its output based on the input data by traversing the network graph and computing each layer. The output is then compared to the expected output using a loss function, which measures the difference between the predicted and actual outputs. In the backward pass, also called backpropagation, the network computes the gradients of the loss function with respect to the weights of each layer in the opposite direction to the forward pass. These gradients are then used to update the network weights and can be synchronized once computed. One forward pass, backward pass, and weight update constitute a training step in which the network gradually learns to minimize the loss function and improve its accuracy on the training data. The main difference between a forward pass and a DNN inference is that a forward pass also requires gradients to be stored.

The number of images of the dataset and the batch size determines the number of training steps of an epoch. We describe the runtime of an epoch as follows:

$$T_{\text{epoch}} = \frac{D}{B \times N} T_{\text{iter}},$$

where D is the dataset size, B is the batch size per computing device, N is the number of computing devices, and T_{iter} is the time to compute a training step. Each computing device computes a mini-batch consisting of B/N elements of the dataset. D/B training steps per epoch are distributed among N computing devices, leading to $D/(BN)$ iterations.

3 APPROACH

Given the computing steps involved in the inference and training phases of deep-learning models that we described in Section 2, we propose a performance model that is simple to build while yielding relatively high accuracy for estimating the training and inference time. In particular, we focus on convolutional neural networks (ConvNets) and include their metrics in the performance model. Moreover, for accurate training-time prediction, we include dataset and training configuration. Such an approach enables generalization to many new unseen models without the necessity to collect a large dataset. It is worth noting that the same analogy can potentially be applied to other deep-learning model categories with minor effort, such as language models, vision transformers, and different training parallelization strategies.

We further use the same parameterized performance model for multiple devices to support a broader range of target platforms, changing only platform-specific coefficients within the performance model. In the following, we describe our performance model for ConvNets.

To estimate the iteration time in the training phase, we decompose an iteration to its most time-consuming steps, namely forward pass, backward pass, and gradient update. Similar to Pei et al. [14], the sum of these three phases yields the total iteration time:

$$T_{\text{iter}} = T_{\text{fwd}} + T_{\text{bwd}} + T_{\text{grad}} \tag{1}$$

The inference time of a deep-learning model is basically a simple forward pass T_{fwd}, whose performance model we describe in Section 3.1. We argue that the time required to run each of the above steps (T_{fwd}, T_{bwd}, and T_{grad}) can be modeled using linear regression given the following ConvNets metrics.

Inputs I: The input tensor size is typically fixed for a given model architecture. The sum of the input tensor size of all convolutional layers represents the memory and processing requirements of the network.

Outputs (or activations) O: The sum of the output tensor size of all convolutional layers. It represents the complexity and total number of features that are extracted by the model from the input data.

FLOPs F: The number of floating point operations of all layers serving as a hardware-independent metric for the computational complexity of a given network.

Weights W: The number of weights determines the memory requirements and the speed of the network.

Layers L: The number of layers of the model, where larger values denote a more complex network with a higher network depth.

Considering that the majority of a ConvNet's runtime is attributed to its convolutional layers, we calculate the inputs and outputs of a ConvNet by parsing its computational graph and summing the metrics for each convolutional layer. Based on the input and output tensor size of each convolutional layer, we compute the FLOPs of a convolution without considering any optimization techniques or actual hardware implementation. Furthermore, inputs, outputs, and FLOPs scale linearly with the batch size. Therefore, we can count these metrics for a single batch size and multiply each with a desired batch size later. This way, recounting these metrics when predicting different batch sizes is not required. Additionally, having the batch size as a parameter in our performance model enables simulating the batch size behavior for larger values, even in scenarios where the batch sizes exceed the memory of a device. Simulating large batch sizes can provide insights into how the training could scale on a different device with more memory or the effects of optimizations such as gradient accumulation.

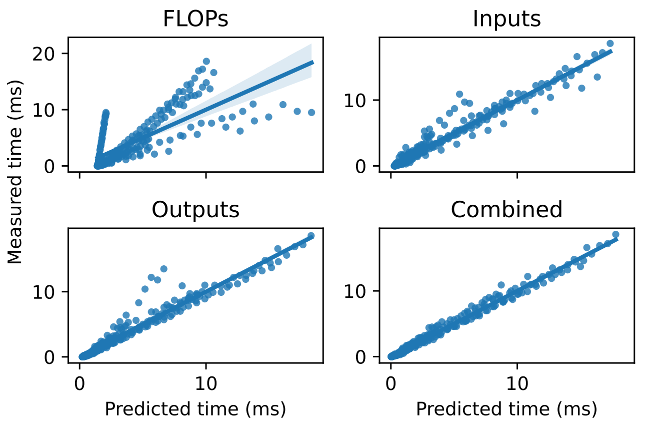

Figure 2: Inference time prediction of various state-of-the-art ConvNets based on FLOPs, inputs, outputs, and their combination. Combining all three metrics leads to the most accurate prediction.

We modeled data parallelism, as it is the primary distribution strategy for ConvNets, but ConvMeter can be extended to support other parallelization strategies, such as model parallelism, by leveraging ConvMeter's capability to predict subgraphs or blocks of DL models, as demonstrated in Section 4.1.2

3.1 Forward pass prediction

Previous work mainly used FLOPs to predict the runtime of ConvNets [17]. However, performance modeling solely based on FLOPS turned out to be an unreliable indicator for inference time. Recent work [4] showed that a ConvNet's outputs (activations) highly correlate with the inference time on memory-bound processors, as the outputs reflect the relative time of storing the results on the device memory. Based on our experiments, we found out that a combination of inputs, outputs, and FLOPs leads to an even more accurate prediction. Combining these metrics incorporates the time it takes to load the inputs of convolutional layers, compute the results, and store them back to the device's memory.

Figure 2 visualizes the predicted inference times based on FLOPs, inputs, outputs, and their combination. We can see that combining the three metrics leads to the most accurate prediction. FLOPs alone are an inadequate predictor of memory-bound processors, as reading and writing the tensors of convolutional layers heavily affects the runtime. Either inputs or outputs alone are also insufficient to predict the inference time, as some models tend to have a disproportionate size of tensors as inputs. The output tensor size of each layer tends to increase throughout most ConvNets. However, DenseNet's input tensor sizes increase while the output tensor remains unchanged within each block. Thus, only considering outputs does not capture such a change in the input tensor size.

The advantage of our analytical model is that hardware details are not incorporated directly into the mathematical expression, and the hardware impact on performance is captured via benchmarking and the coefficients in the regression model. We opted for the linear regression method for simplicity and also due to its reasonably high performance within our context. Unlike DNN-based performance modeling techniques [6, 10, 23], which necessitate extensive training data and numerous epochs to converge, ConvMeter is able to model the performance with reasonable accuracy and much less effort. Our performance model for estimating the forward pass runtime of ConvNets is defined as follows:

$$T_{\text{fwd}} = c_1 \cdot \text{FLOPs} + c_2 \cdot \text{Inputs} + c_3 \cdot \text{Outputs} + c_4, \tag{2}$$

where $c_{1,..,4}$ are tunable platform-specific coefficients. Once the performance model is tuned for a target device, evaluating the term based on the ConvNet metrics yields the forward pass time. As previously mentioned, the ConvNet metrics scale proportionally with the batch size. Therefore, we can add the batch size as a parameter to the performance model by factoring it out from Equation 2.

$$T_{\text{fwd}} = b(c_1 \cdot \text{FLOPs}_{B=1} + c_2 \cdot \text{Inputs}_{B=1} + c_3 \cdot \text{Outputs}_{B=1}) + c_4 \tag{3}$$

The ConvNet metrics Inputs, Outputs, and FLOPs are for batch size equal to one, and $b = B/N$ is the mini-batch size.

Block-wise forward pass time prediction: The same approach for predicting the inference of the entire model also applies to individual blocks within a ConvNet.

As blocks are subsets of neural networks, they are small neural networks themselves, to which we can apply our previously defined inference time performance model.

3.2 Backward-pass prediction

We reuse the same performance model as in the forward pass. Only the platform-specific coefficients need to be tuned based on the measured time of each backward pass. The backward pass tends to take longer as it needs to store the gradient updates during the backward propagation. This additional computation will be captured by performing the linear regression based on the corresponding data from the measured backward pass time. A significant optimization available in Horovod is to start synchronizing the gradient updates during the backward propagation. Instead of waiting until all gradient updates are computed and wasting time, the tensor fusion method synchronizes gradients once they are computed. For only predicting the backward propagation, we can fit the coefficients to Equation 2 based on the backward pass runtimes. However, to predict the entire training time, we must also incorporate the gradient update time, as discussed in the next section.

3.3 Gradient update prediction

We design a performance model for the gradient update using all-reduce synchronization, where a ring-all-reduce pattern synchronizes all local updates. The time to perform this step scales with the number of layers within a model, as frameworks such as Horovod synchronize the gradient updates in a per-layer way. In a distributed training scenario, the gradient update scales with the number of physical nodes in addition to the layers, posing a bottleneck during a network synchronization between computing devices within and between a node. The performance model for the gradient update is, therefore, a linear function that can be fitted using linear regression and has the form:

$$T_{\text{grad}} = \begin{cases} c_1 \cdot L & N = 1 \\ c_1 \cdot L + c_2 \cdot W + c_3 \cdot N & N > 1 \end{cases},$$

where $c_{1,\ldots,3}$ are coefficients that need to be tuned for a target platform. If only one computing device is used for training, the number of layers is sufficient for the gradient update prediction, as the gradient updates are synchronized per layer. To predict the runtime of multiple nodes, we also need to consider the size of the model represented by the number of parameters and the number of nodes, as inter-node communication poses the bottleneck during synchronization.

As mentioned, in practice, the gradient update is not an isolated phase during the training step and overlaps with the backward pass. To capture this behavior, we apply linear regression to our backward pass and gradient update equation combined using the sum of the backward pass and gradient update measurements. For the combined backward pass and gradient update, seven coefficients are required to be fitted based on the data.

3.4 Determining platform-specific coefficients

To determine the coefficients of our inference and training performance model, we first perform hardware benchmarking for various state-of-the-art ConvNet models on a target device to collect the runtime data. Our benchmark runs each selected model with various input image sizes and batch sizes. For the gradient update, we measure the runtime of the same set of ConvNets with different numbers of computing devices and computing nodes. We use linear regression to compute the coefficients for the performance models based on the measurements. For the training performance model, we must compute the coefficients of all three phases summed in Equation 1. For the inference time performance model, we only need to compute the four coefficients of the forward pass in Equation 2.

Our performance modeling method incurs a relatively small overhead, as we only need to compute and store a few coefficients. Moreover, using a simple linear-regression method is faster and more efficient than using complex machine-learning methods used in previous work.

4 EXPERIMENTAL RESULTS

We analyze the accuracy and effectiveness of ConvMeter on various hardware platforms and different conditions. Particularly, we investigate the performance of ConvNets in training and inference phases and demonstrate their scalability to different numbers of nodes and batch sizes. Additionally, we compare ConvMeter with a state-of-the-art prediction model for inference.

Experimental setup

Hardware and software setup. We evaluated our inference prediction on two different platforms containing CPU and GPU processors. The workstation has two Intel Xeon Gold 5318Y processors and four NVIDIA A100 GPUs, each with 80 GB of memory. The HPC cluster comprises GPU nodes, each equipped with two AMD EPYC 7402 processors and four NVIDIA A100 GPUs. Each compute node has four HDR-200 InfiniBand network cards for inter-node communication. We run the DNNs on a single CPU core and a single A100 GPU to evaluate the inference prediction. We use PyTorch for all inference measurements. For single-GPU and distributed training, we deploy Horovod with PyTorch and Adam as the optimizer method. NVIDIA NCCL performs GPU-to-GPU communication within and between computing nodes, utilizing the InfiniBand and NVLink connections on the cluster.

Benchmarks. We measure the inference and training runtime to tune the coefficients of our performance model on each device and collect less than 5,000 different data points for each inference and training scenario. These benchmarks include batch sizes from one to 2048 and image sizes from 32 to 224 pixels, as long as the available memory on the target system allows. We used a wide variety of ConvNet models, ranging from large and generic ones such as AlexNet, VGG [26], ResNets [7], and ResNexts [29] to optimized and mobile-friendly ones, including SqueezeNet [9], MobileNet [8, 22], EfficientNet [27], and RegNets [21]. All models have a distinct architecture, memory usage, and computational intensity, serving as a representative cross-section of most deep convolutional networks. Furthermore, to model the scalability of the training phase in a multi-node distributed environment, we benchmarked the training configuration using different numbers of nodes. In the following sections, we report error statistics for each ConvNet separately. To obtain the error rates per ConvNet, we develop a performance model for each ConvNet, excluding its own data from the training set to ensure unbiased evaluation and accurate prediction accuracy.
Metrics. We assess and report the accuracy of ConvMeter using various metrics, including R^2 and root mean square error (RMSE), normalized root mean square error (NRMSE), and mean absolute percentage error (MAPE). Despite sharing the common goal of assessing the performance of a linear regression model in relation to a given dataset, these key metrics differ in their specific functions. RMSE provides an absolute measure of the accuracy of a regression model's response variable prediction, while R^2 focuses on the extent to which predictor variables explain variability in the response variable. We provide the raw RMSE value as an absolute error rate in addition to a relative RMSE normalized by the range of the data points. On the other hand, MAPE measures the average absolute difference between the predicted and actual values in percent, where large and small errors in the predictions are considered equally important.

In addition to the quantitative metrics above, we provide a scatter plot showing the correlation between prediction and measured values. These plots contain all data points obtained during the evaluation.

Table 1: The correlation and error values of our prediction per ConvNet for a single-CPU and single-GPU inference.

Network	CPU inference			GPU inference		
	RMSE	NRMSE	MAPE	RMSE	NRMSE	MAPE
mobilenet_v2	0.79 s	0.17	0.15	0.08 s	0.16	0.11
resnet18	0.56 s	0.26	0.21	0.04 s	0.14	0.17
resnet50	0.51 s	0.07	0.22	0.09 s	0.12	0.13
resnext50_32x4d	0.69 s	0.08	0.27	0.06 s	0.09	0.10
wide_resnet50_2	0.48 s	0.10	0.15	0.02 s	0.03	0.06
vgg16	0.83 s	0.17	0.18	0.03 s	0.03	0.14
regnet_y_400mf	0.22 s	0.59	0.62	0.15 s	0.27	0.25
squeezenet1_0	0.12 s	0.23	0.52	0.03 s	0.09	0.26
vgg11	0.23 s	0.22	0.21	0.11 s	0.13	0.21

4.1 Inference-phase modeling

ConvMeter is capable of predicting the runtime for the entire model inference and even the constituting blocks of the ConvNets. Such fine-grained runtime information is particularly useful for neural architecture search and network optimization methods to spot and tune the network's bottlenecks [3]. For both aspects, we analyze the accuracy of ConvMeter by comparing the measured inference time on the target device with our prediction. All runtime predictions for a given device use the same coefficients, as we use the same data points from all ConvNets to fit the coefficients. Using all data points ensures a generalized performance model for all ConvNets, capable of predicting new unseen ConvNets without extra tuning steps or limiting our performance model to specific cases (e.g., a fixed batch size).

4.1.1 Entire model time prediction. Figure 3 shows the correlation between actual inference time and prediction on CPU and GPU backends using different batch and image sizes. We achieve the highest prediction accuracy on the CPU backend with R^2 of 0.98, an RMSE of 0.59 s, an NRMSE of 0.13 and a MAPE of 0.25. Notable prediction results on the Nvidia A100 GPU, where R^2 is 0.96, the RMSE is 8.8 ms, the NRMSE is 0.13, and the MAPE is 0.17, suggest a strong predictive performance of the model. Table 1 shows the full breakdown of the evaluated ConvNets. Obtaining such a high prediction accuracy on two different platforms demonstrates that ConvMeter can perfectly capture the essential effects of hardware on the inference time of various deep networks.

We observe that some models, such as AlexNet, have a significantly lower execution time despite the image and batch size due to their lower computational complexity. On the other hand, the ResNet family of models scales well with larger input sizes.

4.1.2 Block-wise time prediction. As mentioned earlier, ConvMeter can be easily adapted to predict the required time to run individual blocks that repeat throughout the structure of deep networks. To demonstrate this feature, we selected a number of blocks from different ConvNets and compared the predicted time with the measured runtimes. Table 2 lists the blocks used and the corresponding ConvNets they are from. The implementation of these blocks is from the ConvNets available in Torchvision 0.14.0.

Figure 4 visualizes the relation between measured block runtime and our prediction. The prediction achieves a correlation $R^2 = 0.997$, an RMSE of 0.67 ms, an NRMSE of 0.15, and a MAPE of 0.16. Table 2

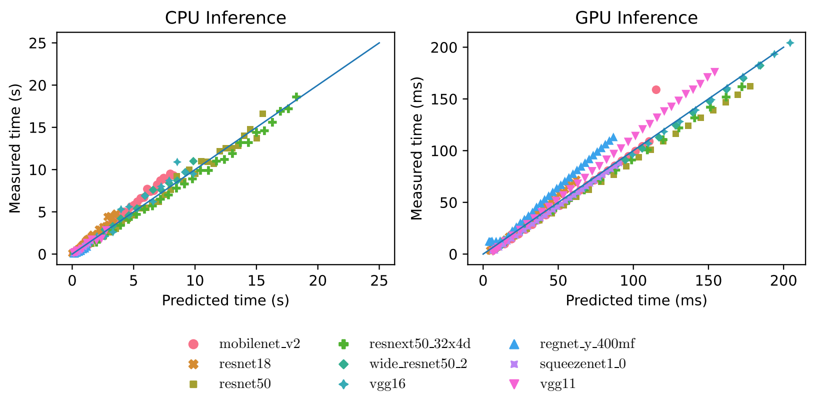

Figure 3: Prediction accuracy of the inference time on Intel Xeon CPUs and Nvidia A100 GPUs.

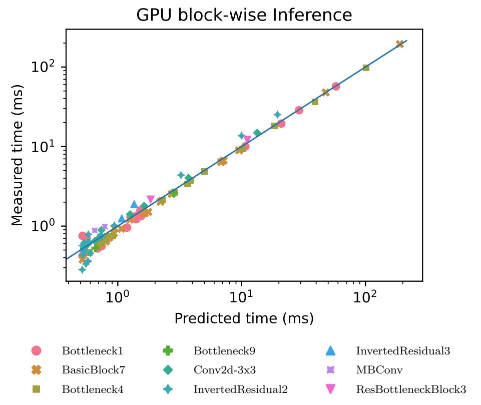

Figure 4: Prediction accuracy of the block-wise inference time on A100 80GB.

Table 2: The correlation and error values of our inference time prediction for each block.

Network		Metrics		
Block	Model source	RMSE	NRMSE	MAPE
Bottleneck1	ResNeXt50-32x4d	1.87 ms	0.24	0.10
Bottleneck4	ResNet50	1.03 ms	0.12	0.10
Conv2d_3x3	InceptionV3	1.47 ms	0.29	0.10
BasicBlock7	ResNet18	0.42 ms	0.1	0.16
InvertedResidual2	MobileNetV3	0.59 ms	0.49	0.2
ResBottleneckBlock3	RegNet-X-8gf	0.31 ms	0.05	0.09
Bottleneck9	Wide-ResNet50	0.16 ms	0.06	0.14
MBConv	EfficientNet-B0	0.28 ms	0.30	0.3
InvertedResidual3	MobileNetV2	0.32 ms	0.26	0.37

Table 3: The correlation and error values of our prediction per ConvNet for a single-GPU scenario and distributed training on multiple nodes.

Network	Single-GPU training			Distributed training		
	RMSE	NRMSE	MAPE	RMSE	NRMSE	MAPE
vgg16	44.21 ms	0.28	0.18	51.7 ms	0.16	0.17
efficientnet_b0	27.3 ms	0.22	0.15	29.88 ms	0.15	0.12
regnet_y_400mf	19.41 ms	0.14	0.13	38.57 ms	0.18	0.15
squeezenet1_0	10.1 ms	0.14	0.17	24.98 ms	0.18	0.13
alexnet	42.64 ms	0.44	0.22	48.9 ms	0.21	0.17
vgg11	29.64 ms	0.27	0.28	38.61 ms	0.14	0.11
resnet18	17.41 ms	0.19	0.15	32.58 ms	0.21	0.17
resnet50	25.35 ms	0.21	0.21	25.01 ms	0.13	0.11

lists the correlation and error values for each tested block. The maximum MAPE is 0.37 for MobileNet. Except for MobileNets and EfficientNet, our prediction achieves MAPE error rates from 0.09 to 0.2. The accurate prediction of this model makes it suitable for estimating the performance of single blocks during the design process of ConvNets, such as in neural architecture search [3].

4.1.3 Comparison to SotA. We compared the inference prediction accuracy of ConvMeter to that of the state-of-the-art DIPPM [23], which we chose due to its recent development and its specific dataset for A100 GPUs, identical to those used in our experiments. We employed a fixed image size of 128x128 pixels and varied batch sizes from 16 to 2,000. The results are presented in Figure 6, where the mean absolute percentage error (MAPE) and normalized root mean square error (NRMSE) metrics are reported for both models. DIPPM was unable to parse the model graph of squeezenet1_0, making a comparison impossible for this architecture. ConvMeter outperforms DIPPM across all scenarios, demonstrating improved inference prediction accuracy.

4.2 Training-phase modeling

We evaluated the prediction for single-GPU training on a workstation using an Nvidia A100 GPU for a training step. To predict the epoch time and the entire training time, we first need to predict the time required for a training step and multiply this value by the number of training steps. This factor is fixed and can be computed using the dataset size, batch size, and the number of computing devices as discussed in Section 3. In the following, we present the training step time prediction because the entire training and epoch time can directly be derived from that.

In Figure 5, we see minimal variation for the same ConvNet with a MAPE of less than 0.28, as no network communication is involved while updating the gradients. There is a higher deviation of the forward pass prediction for low runtimes, mainly caused by running small models with small batch and image sizes, resulting in a low computational intensity and underutilization of the A100. However, the prediction is more accurate for larger batch sizes, which is usually desired during training. In total, our prediction achieves an R^2 of 0.88, an RMSE of 29.38 ms, an NRMSE of 0.26, and a MAPE of 0.18. The error values for each model are listed in Table 3.

4.2.1 Distributed Training. To evaluate the prediction on multiple nodes, we performed the same measurements with various ConvNets and hyperparameters on a different number of nodes. As shown in Figure 7, the training spends most of its time during the backward pass and gradient update. The runtime of the backward pass and gradient update is in the same plot, as both phases overlap. However, network communication causes much more variance for the same training configuration, leading to more variance in

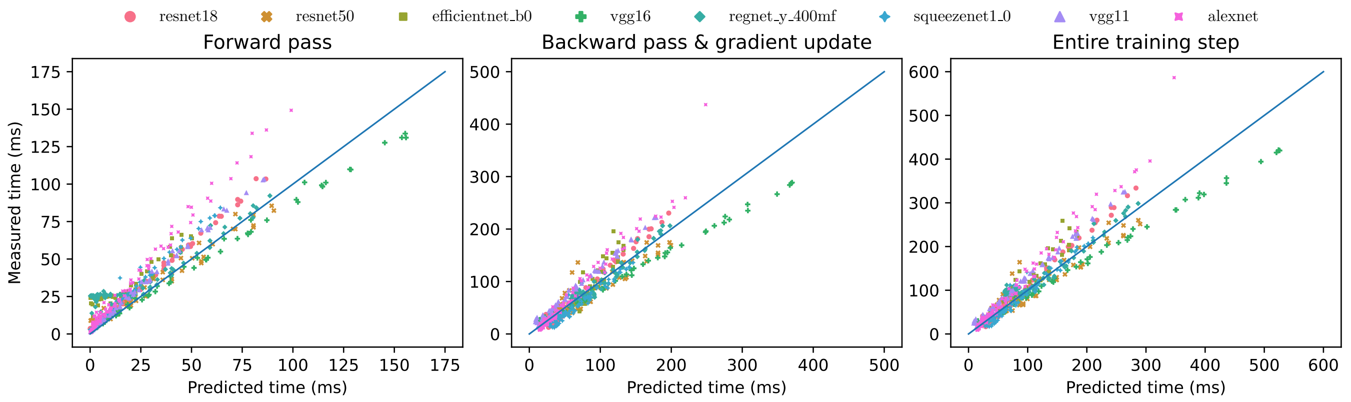

Figure 5: Prediction accuracy of the three phases of a DL training step and the entire training step for various models on a single A100 GPU. The entire training step is a combination of the forward pass, backward pass, and gradient update.

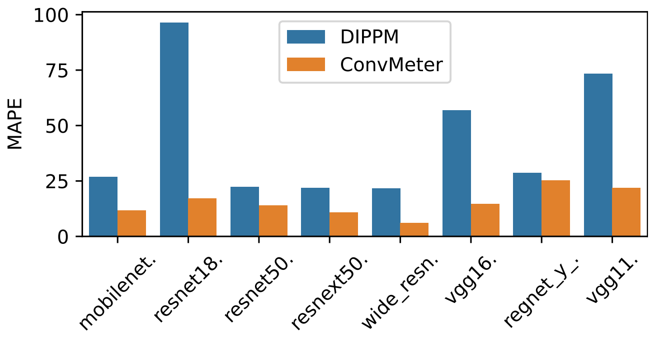

Figure 6: MAPE error rates of ConvMeter (orange) compared to the prediction model DIPPM [23] (blue). See Table 1 for the full ConvNet names.

the measured data and, therefore, a less accurate prediction. We see more variation for the forward and backward pass around the regression line than for the single-GPU training. Although both phases do not require synchronization, the synchronization from the gradient update affects the runtime of these phases, as not all devices simultaneously start their phases after the gradient update.

Similar to the single-GPU training, the prediction is more accurate for larger image sizes and batch sizes, as they utilize the devices more effectively. ConvMeter is much more accurate in predicting the forward and backward pass, as no network communication is involved, leading to less variance in the measured data. The gradient update phase mainly scales with the model size and the allocated nodes, leading to a higher computational overhead for smaller batch sizes. Therefore, users typically maximize the per-device batch size in distributed training before expanding the training to more GPUs to reduce communication overhead. In other words, adding more nodes to training while keeping the batch size unchanged increases the communication overhead. Thus, we are mainly interested in predicting larger batch sizes, and our prediction gets more accurate as the batch size increases. Despite the considerable variance of the gradient update prediction, the overall training step prediction for all batch sizes achieves an R^2 of 0.78, an RMSE of 38.71 ms, an NRMSE of 0.18, and a MAPE of 0.15. The exact numbers per model are listed in Table 3.

4.3 Scalability analysis

ConvMeter can predict the runtime based on the number of computing nodes. We can use this information to determine the turning point when additional computing nodes do not lead to a significant reduction in the training time. The increasing network overhead and fixed computational load per computing node typically lead to this diminishing return, but its turning point differs between different DNNs and training parameters.

Figure 8 shows the scalability of different ConvNets for different numbers of computing nodes with a fixed image size of 128×128 px and batch size of 64.

For this experiment, we measured the training time of eight ConvNets with different numbers of nodes, batch, and image sizes. Following the same approach as before, the ConvNet for each performance model shown in Figure 8 was not present in the training data set and is therefore unknown to the performance model. Although we can observe some deviation from the measurements, our prediction follows the same trend and preserves the scalability of each ConvNet, also showing our performance model's ability to handle noise in the measured data. While most models follow a more steep trajectory, Alexnet shows a more prominent diminishing return, which our prediction correctly reflects. We believe that the error rates reported for the training time prediction are reasonably useful within the context of training infrastructure planning or task schedulers.

As explained in Section 3, ConvMeter has a parameter for the batch sizes. This parameter enables the inference and training runtime prediction of different batch sizes. Another hyperparameter called learning rate is usually adjusted with the batch size, which affects the time until the model converges. The learning rate is multiplied in every iteration regardless of its value and does not affect the epoch time and is, therefore, not included in our performance model. We can predict the runtime even for batch sizes that would exceed the capacity of the training device. Simulating larger batch sizes can be valuable information for scheduling and

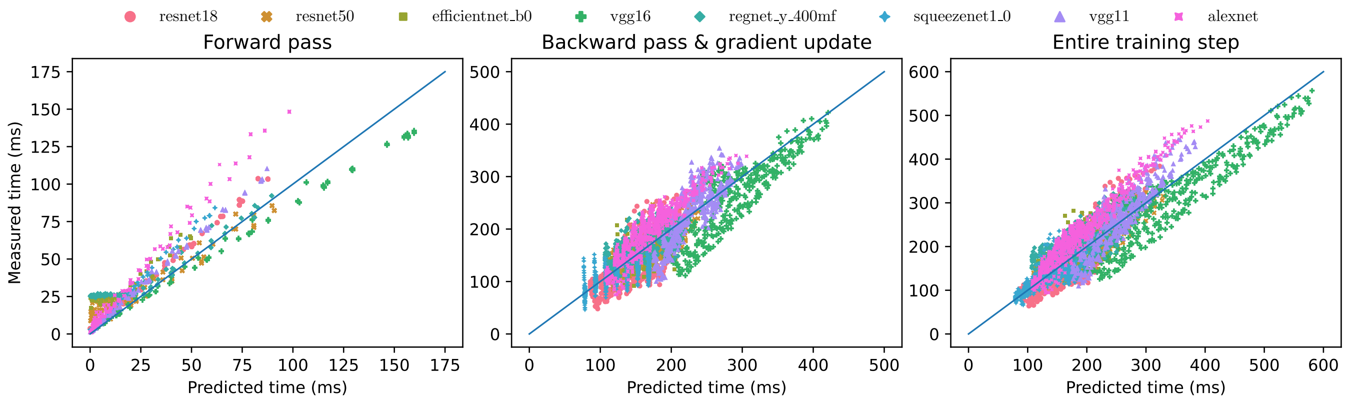

Figure 7: Prediction accuracy of the three phases of a training step and the entire training step for various models on multiple A100 nodes. The entire training step is a combination of the forward pass, backward pass, and gradient update.

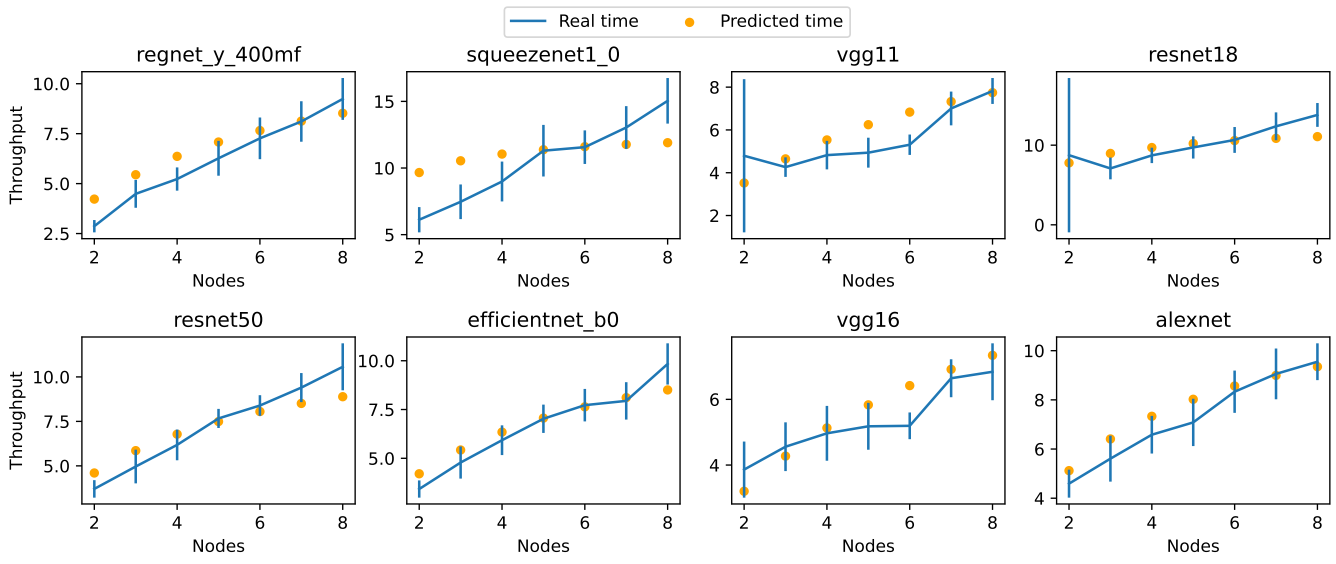

Figure 8: Throughput (images per second) prediction for different numbers of nodes and fixed batch size and image size. The blue vertical lines show the standard deviation of the measured data.

potential hardware upgrades, as main memory is a common bottleneck in DNN training. Suppose we are interested in the scalability of known models instead of predicting the runtime of unknown models. In that case, we can tune the coefficients based on a specific ConvNet of interest to predict its scalability more accurately. We do not need to rerun benchmarks and can reuse the data and apply the regression on the specific ConvNet instead of data points from all ConvNets. Since our prediction works with a variable number of nodes and batch sizes, we can predict both weak scaling and strong scaling, as our performance model can predict the scaling behavior of nodes for a fixed global batch size and the scaling behavior of nodes for an increasing global batch size.

Figure 9 highlights that ConvMeter can predict how the training time scales for a given model and batch size. This model can predict runtime for any batch size, extending beyond the batch sizes included in our benchmark dataset, and is applicable even for models not previously encountered. Additionally, the figure reveals that the impact of increasing the batch size on throughput is not uniform across different models. While most models exhibit good scalability up to a batch size of 2048, both ResNet18 and SqueezeNet demonstrate a more pronounced diminishing return at larger batch sizes. Each model exhibits varying scalability [25] when it comes to the impact of increasing batch size on training time, and our prediction accurately captures this phenomenon.

5 RELATED WORK

Among the existing methods for performance prediction of DNNs, most of them focus on predicting the inference or training time

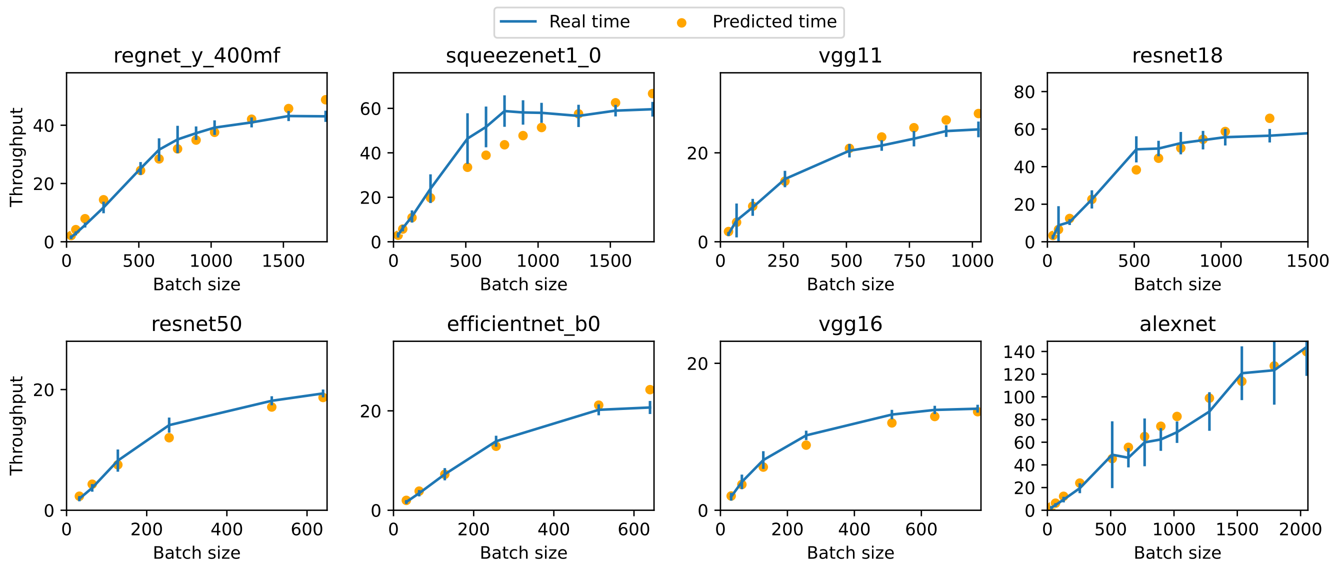

Figure 9: Throughput (images per second) prediction for each model using a fixed image size when increasing the batch size. The blue vertical lines show the standard deviation of the measured data.

within a single computing node. Our work has surpassed this limitation and is capable of providing accurate predictions for distributed environments as well. As a big portion of the research has only focused on inference-phase time prediction, we categorize the existing work based on its operation scope, either inference time alone or inference time with or without training time. In contrast to most related work, our work supports both scopes. Based on the available information, we summarized the related work according to their features in Table 4.

Inference-phase performance models. NeuralPower [1] estimates the inference time, power, and energy consumption of ConvNets. However, it was designed for simple architectures such as AlexNet [12] and VGG [26] and does not cover more complex and modern structures such as ResNet [7]. nn-Meter [32] predicts the inference time of ConvNets by identifying the kernels running on a target device during inference and building a prediction model based on each kernel runtime estimation. However, nn-Meter requires a lot of sampling data to work well and only estimates the inference. DIPPM [23] predicts the inference latency, energy, and memory usage of DL models by analyzing their graph structure. Their performance model achieves very low error rates but requires 500 epochs of training on a large dataset. We compared our prediction model to DIPPM in Section 4.1.3.

Training-phase performance models. Justus et al. [10] trained a deep-learning model to predict the inference and epoch time of DNNs. However, they only focused on single-device training. Pei et al. [14] developed a model to predict multi-GPU training, but only on a single node. PALEO [17] follows a similar analytical approach, decomposing the runtime of each layer in reading the inputs, calculating the results, and writing the results. However, it estimates the runtime of each phase by dividing the load (data to read/write, FLOPs to compute) by the relative performance of the device. As we discussed earlier, only using the FLOPs does not reflect the complex structures of modern ConvNets. ParaDL [11] predicts the performance of a ConvNet training workload for different deep-learning parallelization techniques and find bottlenecks. However, the prediction will be confined to the given model and cannot predict unseen models. Habitat [31] extends the performance prediction of DNNs from an existing hardware setup to a new, different hardware configuration. Habitat predicts the iteration time of DL training, but lacks the capability to predict times for distributed training, and is constrained to the specific batch size it was trained on. DNNPerf [6] is another ML-based tool to predict the training time and GPU memory consumption of DNNs. Although it can predict unseen models, it is tuned for a single computing node and, therefore, cannot model distributed training.

In contrast to the methods above, our performance model can predict the inference time of unseen models, constituent blocks, and the training time on single GPUs and distributed systems. We can further predict the scalability based on the batch size, one of the most important hyperparameter for scaling deep-learning training.

6 CONCLUSION AND OUTLOOK

Performance modeling of DNNs has shown promising results in various applications, such as infrastructure planning and DNN design optimization. In this paper, we proposed ConvMeter, a simple yet effective performance model to predict the inference and training time of ConvNets. We demonstrated the possibility of predicting the runtime of ConvNets without complex machine-learning models or performance models that require extensive sampling and multiple passes of fine tuning. Such complex approaches incur a relatively large overhead in generating the prediction model, as some require iterative tuning through adaptation phases [32] or

Table 4: Summarized comparison of our work with state-of-the-art methods. The column "Modeling effort" lists the required data points, epoch time, or the training time to generate the performance model, if available.

Method	Modelling technique	Modeling effort	Inference prediction		Training prediction			
			Block-wise	Entire model	Single GPU	Multiple GPUs	Multiple nodes	Scalability analysis
NeuralPower [1]	Regression	N/A	–	✓	–	–	–	–
nn-Meter [32]	Regression	1-4 days	–	✓	–	–	–	–
DIPPM [23]	MLP	200,000 points, 500 epochs	–	✓	–	–	–	–
DNNPerf [6]	MLP	200 epochs	–	–	✓	✓	–	–
Habitat [31]	Wave scaling, MLP	N/A	–	–	✓	–	–	–
Justus et al. [10]	MLP	50,000 points, 300 epochs	–	–	✓	✓	–	–
Pei et al. [14]	Analytical model	N/A	✓	✓	✓	✓	–	–
PALEO [17]	Analytical model	N/A	–	✓	✓	✓	✓	–
ConvMeter	Regression	<5,000 data points	✓	✓	✓	✓	✓	✓

training a prediction model for hundreds of epochs [23]. In contrast, our simpler approach only requires linear regression, and building the performance model is significantly faster. ConvMeter exploits the characteristics of ConvNets and the training scenario to estimate the inference time and the main steps involved in the training phase. Besides predicting the inference time of the entire ConvNet, we demonstrated that ConvMeter can also predict the runtime of constituent ConvNet blocks, which is helpful in the ConvNet design process. Furthermore, we analyzed the scalability of various ConvNets in terms of computing nodes and input batch size using our performance model.

In future work, we aim to analyze other DNNs, such as language models and vision transformers, which are the current mainstream deep-learning models in the research community, and different parallelization strategies for distributed training. Moreover, we aim to study edge processors and analyze our performance model on such systems with limited resources.

ACKNOWLEDGMENTS

The German Federal Ministry of Education and Research (BMBF) and the Hessian Ministry of Science and Research, Art and Culture (HMWK) supported our work as part of the NHR funding. We also appreciate the computing time for the high-performance computer Lichtenberg at the NHR Center NHR4CES@TUDa, which is financially supported by BMBF and HMWK. Furthermore, we would like to thank the Gauss Centre for Supercomputing e.V. for providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC). In addition, we acknowledge the funding received from BMBF Software Campus Program and the German Research Foundation (DFG, Project No. 449683531).

REFERENCES

[1] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. 2017. NeuralPower: Predict and Deploy Energy-Efficient Convolutional Neural Networks. *CoRR* abs/1710.05420 (2017). arXiv:1710.05420

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *Proc. of International Conference on Learning Representations (ICLR).*

[3] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *Proc. of International Conference on Learning Representations (ICLR).*

[4] Piotr Dollár, Mannat Singh, and Ross B. Girshick. 2021. Fast and Accurate Model Scaling. *CoRR* abs/2103.06877 (2021). arXiv:2103.06877

[5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research* 20, 1 (jan 2019), 1997–2017.

[6] Yanjie Gao, Xianyu Gu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. 2023. Runtime Performance Prediction for Deep Learning Models with Graph Neural Network. In *Proc. of 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* 368–380.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385

[8] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. *CoRR* abs/1905.02244 (2019). arXiv:1905.02244

[9] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR* abs/1602.07360 (2016). arXiv:1602.07360

[10] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. 2018. Predicting the Computational Cost of Deep Learning Models. *CoRR* abs/1811.11880 (2018). arXiv:1811.11880

[11] Albert Njoroge Kahira, Truong Thao Nguyen, Leonardo Bautista Gomez, Ryousei Takano, Rosa M. Badia, and Mohamed Wahib. 2021. An Oracle for Guiding Large-Scale Model/Hybrid Parallel Training of Convolutional Neural Networks. In *Proc. of the International Symposium on High-Performance Parallel and Distributed Computing.* 161–173.

[12] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *CoRR* abs/1404.5997 (2014). arXiv:1404.5997

[13] J Gregory Pauloski, Lei Huang, Weijia Xu, Kyle Chard, Ian T Foster, and Zhao Zhang. 2022. Deep Neural Network Training With Distributed K-FAC. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 3616–3627.

[14] Ziqian Pei, Chensheng Li, Xiaowei Qin, Xiaohui Chen, and Guo Wei. 2019. Iteration Time Prediction for CNN in Multi-GPU Platform: Modeling and Analysis. *IEEE Access* 7 (2019), 64788–64797.

[15] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chen Meng, and Wei Lin. 2021. DL2: A Deep Learning-Driven Scheduler for Deep Learning Clusters. *IEEE Transactions on Parallel and Distributed Systems* 32, 8 (2021), 1947–1960. https://doi.org/10.1109/TPDS.2021.3052895

[16] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *Proc. of International Conference on Machine Learning (ICML).* PMLR, 4095–4104.

[17] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. Paleo: A Performance Model for Deep Neural Networks. In *Proc. of International Conference on Learning Representations (ICLR).*

[18] Aurick Qiao, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. 2020. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. *CoRR* abs/2008.12260 (2020). arXiv:2008.12260

[19] Ilija Radosavovic, Justin Johnson, Saining Xie Wan-Yen Lo, and Piotr Dollár. 2019. On Network Design Spaces for Visual Recognition. In *Proc. of International Conference on Computer Vision (ICCV).*

[20] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing Network Design Spaces. In *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[21] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2020. Designing Network Design Spaces. *CoRR* abs/2003.13678 (2020). arXiv:2003.13678

[22] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *CoRR* abs/1801.04381 (2018). arXiv:1801.04381

[23] Panner Selvam, Karthick, and Mats Brorsson. 2023. DIPPM: A Deep Learning Inference Performance Predictive Model Using Graph Neural Networks. In *Proc. of Euro-Par 2023: Parallel Processing*, José Cano, Marios D. Dikaiakos, George A. Papadopoulos, Miquel Pericàs, and Rizos Sakellariou (Eds.). 3–16.

[24] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[25] Christopher J. Shallue, Jaehoon Lee, Joseph M. Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. 2018. Measuring the Effects of Data Parallelism on Neural Network Training. *CoRR* abs/1811.03600 (2018).

[26] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proc. of 3rd International Conference on Learning Representations (ICLR)*, Yoshua Bengio and Yann LeCun (Eds.).

[27] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *CoRR* abs/1905.11946 (2019). arXiv:1905.11946

[28] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2018. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. *CoRR* abs/1812.03443 (2018). arXiv:1812.03443

[29] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2016. Aggregated Residual Transformations for Deep Neural Networks. *CoRR* abs/1611.05431 (2016). arXiv:1611.05431

[30] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. 2021. Computational Performance Predictions for Deep Neural Network Training: A Runtime-Based Approach. *CoRR* abs/2102.00527 (2021). arXiv:2102.00527

[31] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. 2021. Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In *Proc. of 2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 503–521.

[32] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices. In *Proc. of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, New York, NY, USA, 81–93.