IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 15 May 2023, accepted 4 June 2023, date of publication 20 June 2023, date of current version 26 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3287998

RESEARCH ARTICLE

A Runtime Switchable Multi-Phase Convolutional Neural Network for Resource-Constrained Systems

JEONGGYU JANG[1] AND HOESEOK YANG[2], (Member, IEEE)
[1]Department of Electrical and Computer Engineering, Ajou University, Suwon-si 16499, South Korea
[2]Department of Electrical and Computer Engineering, Santa Clara University, Santa Clara, CA 95053, USA

Corresponding author: Hoeseok Yang (hoeseok.yang@scu.edu)

This work was supported by the Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) through the project ''Neuromorphic Computing Software Platform for Artificial Intelligence Systems'' under Grant 1711080972.

ABSTRACT Convolutional Neural Networks (CNNs) are widely used in various systems, including resource-constrained embedded systems or IoT devices. In such systems, it is typical to deploy compressed or pruned CNNs, instead of original ones, at the cost of reduced accuracy. Existing CNN pruning techniques have primarily focused on minimizing resource requirements. However, today's embedded systems are increasingly dynamic in both resource demands and availability. Thus, the previous techniques that only consider given static cases are no longer efficient. In this paper, we propose a novel multi-phase CNN that enables a multi-objective exploration of a number of pruning candidates out of a single CNN. In the proposed technique, a CNN can operate in various versions depending on which subsets of weights are used and can be transformed to the one best matches to the given constraint adaptively and efficiently. For that, a CNN is first pruned to the sparsest form; then a set of parameters (sub-network) is additionally supplemented as the phase goes by. As a result, a number of network versions for all different phases can be represented by a single network and they form a pareto solution over the accuracy and resource usage trade-off. In this work, we target CPU-based CNN inference engines as most embedded systems do not have the luxury of specialized co-processor support such as GPUs or HW accelerators. The proposed technique has been implemented in a publicly available CPU inference engine, Darknet, and its effectiveness has been validated with a popular CNN in terms of design space exploration capability and runtime switchability.

INDEX TERMS Deep learning, convolutional neural network, neural network optimization, resource-constrained system.

I. INTRODUCTION

The last decade has witnessed a dramatic growth of convolutional neural network (CNN) applications in image classification and recognition [1], [2], [3], semantic segmentation [4], [5], image enhancement [6], [7], and so forth. Such successes have been attributed to the enhanced accuracy enabled by a large number of layers cascaded in a row in a single CNN, so-called deep learning, which, in turn, owes to the advance of parallel computer architectures [8], [9] and CNN accelerators [10], [11].

The associate editor coordinating the review of this manuscript and approving it for publication was Jie Tang.

Nowadays, the application of deep CNNs is continuously becoming more pervasive in daily lives. So, there are increasing demands to successfully operate CNNs on embedded systems or IoT (Internet-of-Things) devices, where only limited resources are available. Two different approaches have been proposed to overcome the insufficient resource availability in such systems. The first approach is to use reduced or simplified models in the first place [12], [13] that require a smaller storage or memory to store parameters, and lesser computing capability. The other is to compress the CNN model by pruning out less important weights [14], [15], [16], [17], [18], [19].

Designing an electronic system is often associated with multiple non-functional design concerns, e.g., power

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

consumption, memory capacity, latency constraint, and so forth. That is, in many cases, more than one optimization objectives should be considered with a number of resource constraints. Moreover, as those factors are closely related to each other, it is crucial to have a systematic methodology to consider them altogether in the design and optimization phases [20], [21], [22], [23], [24], [25]. Most CNN optimization methods [12], [13], [14], [15], [16], [17], [18], [19] are typically proposed to explore the trade-off between accuracy and resource usage.

In addition, today's embedded systems are increasingly dynamic in terms of the above-mentioned design concerns. On one hand, the execution time demand may vary due to the multiple operation modes [26] or dynamic input workload characteristics that is associated with dynamic physical processes [27]. On the other hand, resource availability can also be flexible as in the transiently powered systems with energy harvester where power budget may change at runtime [28]. Moreover, CNN workloads often exhibit different context-/scene-specific behaviors [29] or quality requirements [30]. In short, CNNs running on embedded systems need to adapt themselves efficiently in response to the dynamic context or workload changes, which is the main focus of this paper.

It is not a viable solution to keep multiple versions of a CNN due to the limited storage capacity of embedded systems. Alternatively, an approach, which we refer to as *multi-phase* or *N-in-1* network in this paper, has been proposed [16], [19]. In these works, a single CNN can be trained to be used in many different forms. In these techniques, while only one set of CNN parameters is maintained in the system, different subsets of it may be activated at runtime.

Tann et al. [19] proposed an *incremental* training methodology to build a multi-phase CNN. In their work, they first obtain the most compact form of CNN out of the original network by removing a number of channels and kernels in each layer. Then, based on this *fixed* CNN, they add new channels and kernels to each layer and only the weights that belong to the newly added channels and kernels go through a training procedure again. By repeating such *incremental* training procedures, they could obtain an N-in-1 network, with which one can selectively activate or deactivate a subset of channels and kernels in the CNN. Another approach to obtain an N-in-1 network is *NestedNet*, proposed by Kim et al. [16], that requires only one training step. They define all possible multi-phase configurations of a CNN in advance. Then, in the training stage, the losses of all configurations are separately calculated and a weighted sum of them is used as the global loss for back propagation.

We pay attention to the following limitations of existing multi-phase CNN approaches. First one is on the granularity of building N-in-1 networks. It is well-known that the fine-grain weight pruning approach such as [14] and [15] is not suitable to achieve speedup on commodity hardware like CPU or GPU, but can only effectively improve the inference speed on top of specially designed HW accelerators [10], [11], [31], [32]. That is, removing weights at arbitrary positions would not improve the inference speed. For the same reason, existing N-in-1 CNN approaches either restricted themselves to the coarse-grain structured granularity [19], i.e., filter- or kernel-level, in building the multi-phase CNN or cannot expect the inference speed gain from a smaller subset of the multi-phase CNN when using the weight level granularity [16], [17]. As most commodity hardware of embedded systems do not have the luxury of the customized accelerators, the multi-phase CNN should be effectively working on top of CPU-based hardware. On the contrary, the proposed multi-phase CNN approach makes use of two different granularities in order to enable more efficient exploration of the speed-accuracy trade-off for CPU-based embedded systems. The second limitation is the runtime adaptability. None of the existing multi-phase CNN approaches mentioned above considered efficient runtime switching to the best of our knowledge.

Considering the limitations reviewed above, we argue that the following challenges need to be addressed to enable the efficient runtime reconfiguration of N-in-1 CNN on embedded systems. First, multi-phase with a fine granularity that is tailored to the underlying CPU architecture is indispensable. This is critical to enable a sophisticated design space exploration over the speed-accuracy trade-off. Second, in order to effectively reduce the memory usage, a coarse-grain multi-phase approach, in which a substantial volume of spatially co-located weights can be added or removed at the same time, should be combined with the fine-grain one. Lastly, the inference engine needs to be properly extended to facilitate prompt runtime switching of N-in-1 network without considerable switching delay.

In this paper, we propose a multi-phase CNN that overcomes the above challenges tailored to embedded microprocessors. To be more specific, training is performed in an iterative way at two different levels. At fine-grain, CNN weights are removed or added in an architecture-specific manner considering the SIMD (Single-Instruction-Multiple-Data) width of the target CPU. A traditional coarse-grain approach, also known as *structured sparsity* [33], is combined with the fine-grain one to enable more comprehensive design space explorations. Runtime switching of the proposed multi-phase CNN is implemented and evaluated on top of a publicly available inference engine for CPU, Darknet [34].

The contribution of this paper can be summarized as follows:

- We propose a novel multi-phase CNN pruning technique that enables an efficient runtime switching between variable candidates from a single CNN.
- In doing so, a two-level architecture-aware multi-phase CNN building method, with respect to a number of non-functional design concerns like latency or power consumption, is devised to determine suitable subsets of weights to be activated.

- Unlike other existing approaches, the proposed technique considers the actual memory usage at runtime. For that, the inference engine is properly extended to support runtime partial loading of CNN weights.

The rest of this paper is organized as follows: In the following section, existing approaches are reviewed in their key ideas and limitations. In particular, we will describe the advantages and disadvantages of the existing methods focusing on the accuracy-resource trade-off. Furthermore, we present an overview of previous studies on the multi-phase CNN. In Section III, we elucidate the iterative training method employed to construct the proposed multi-phase CNN, followed by an exploration of the two architecture-aware sparsity levels used in pruning/restoring process in Section IV. Subsequently, we discuss the procedure of building a specific multi-phase CNN with respect to the given design constraints in Section V. In Section VI, we demonstrate the effectiveness of the proposed technique through experimental results. Finally, we conclude the paper with a brief summary in Section VII.

II. RELATED WORK

Han et al. [14], [15] proposed to optimize CNNs by pruning out less important weights, so-called weight-level pruning. While the gain in model size was evidently substantial and measurable in this technique, its impact on the latency, i.e. inference speed, was just optimistically predicted in terms of number of operations (FLOPS). To achieve actual gain in inference speed, CNNs should be executed on top of a special inference accelerator [11], called *EIE*, where only non-zero weights are autonomously detected and computed from the sparse matrix. Albericio et al. [35] also proposed a novel architecture that can eliminate ineffectual multiplication caused by sparse matrices.

To overcome this limitation of weight-level pruning, it has been proposed to perform pruning at a coarser granularity [33], i.e., filters or kernels are pruned in a bulky manner. As the matrices to be computed structurally shrink, the inference speed could be directly accelerated without special HW support as the degree of pruning increases. However, in this approach, such coarse granularity greatly impairs the degree of freedom in pruning, and thus may cause non-negligible accuracy loss.

Yu et al. [36] proposed a fine-grain pruning technique where the pruning unit is a group of consecutive weights whose size is as big as the SIMD width of the underlying microprocessor. Their approach is similar to ours in that the pruning is performed in an architecture-aware manner. However, since they rely on a certain sparse matrix multiplication, the pruning improves the inference speed only in case of high sparsity.[1] In contrast, the proposed technique is based on the GEMM (General Matrix Multiply) which is widely used in modern deep learning inference engines, making it more generically applicable and useful in any degree of sparsity.

A number of multi-phase CNN techniques have been proposed. Tann et al. [19] proposed a framework that can explore step-wise energy-accuracy trade-off. It can be regarded as a general multi-objective optimization technique as it is capable of finding a pareto-front solution over the two objectives, energy and accuracy. Throughout its multiple *incremental training* phases, each layer of the target CNN is complemented with additional channels. Amir and Givargis [17] also proposed a similar approach, called *priority neuron network*, for resource-aware optimization of CNNs. In this framework, the importance of the neurons are trained together with weight parameters at the same time. Like [19], it only needs to maintain a single set of weights and is still capable of selectively choosing a subset of weights to participate in the inference procedure by changing the priority parameter. Similarly, Kim et al. [16] propose to build a multi-phase CNN from a single CNN in their framework called *NestedNet*. They obtained a subset CNN by applying pruning at three different levels: layer, channel, and weight. While all the three levels have direct impact on model size and accuracy, it has been reported that weight-level pruning shows no significant impact on the inference speed. Yu et al. [37] also proposed a multi-phase CNN technique, called *Slimmable Neural network*. Unlike the other techniques previously mentioned, they constructed a multi-phase CNN only by using switchable batch normalization.

Our approach is similar to the above mentioned multi-phase CNN techniques in the sense that the training is performed over the multiple phases and the weights can be selectively added (or excluded) to (from) the basic network. However, as their approach either only took the coarse-grain sparsity, i.e., entire channels or layers, into consideration [19], [37] or could not actually take advantage of the benefit of the fine-grain pruning in inference speed [16], [17], they could not comprehensively explore the design space over the resource-accuracy trade-off. In contrast, we take full advantage of both coarse- and fine-grain sparsity levels tailored to the underlying hardware architecture, i.e., SIMD width or memory capacity. Another important difference of the proposed technique is on the runtime switchability. While only [19] among the above mentioned techniques mentioned the runtime reconfiguration, no detailed switching principle has been reported. We show in this work how the proposed technique enables better design space exploration over the multiple design concerns, compared to the existing techniques, with the runtime switching capability.

III. MULTI-PHASE TRAINING

A. ITERATIVE MULTI-PHASE TRAINING

In multi-objective optimization, there may be a number of optimal design candidates, each of which is not dominated by the others, over the Pareto front of the multiple design concerns. A typical example of such multi-objective

[1] In the matrix-vector operation of fully connected layers, their matrix multiplication is beneficial in speed when the sparsity is more than 3%. On the other hand, in the matrix-matrix operation of convolution layers, the speed gain is only achieved when the sparsity is above 83% [36].

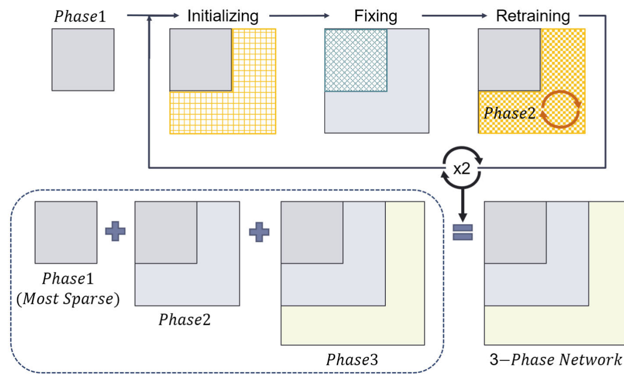


FIGURE 1. A 3-phase training example of the proposed multi-phase training method.

optimization in CNN is the exploration of the accuracy-parameter trade-off. There are a number of approaches that explore this multidimensional design space by applying the multi-phase training approach to a single CNN, as discussed in the previous section [16], [17], [19].

In a multi-phase CNN, all parameters of the original CNN are partitioned into several distinct subsets. Among them, a core subset, which is the smallest phase, is always activated for each phase. Then, in addition to this core subset, other subsets can be added one by one as needed to improve the inference accuracy. In this way, it is possible to effectively operate several different CNN versions with different levels of accuracy, while using only as much memory space as a single original CNN. Figure 1 illustrates how a 3-phase CNN can be obtained by applying an iterative training method. First, the original CNN can be transformed to the core subset (*Phase1*) by pruning. Then, retraining is performed by adding additional parameters (*Phase2*) to this core subset. Note that the already existing parameters remain fixed in this retraining phase. This procedure is repeated until the desired number of phases is obtained.

How to determine the separation of the parameter subsets and how to train each of them differs from one approach to another. Tann et al. [19] only considered coarse-grain parameter units, i.e., kernels or channels, for the parameters to be added at each phase. From the sparsest version, a set of channels is added in each phase, and how much to be added is decided iteratively based on the score margin. Kim et al. [16] and Amir and Givargis [17] did not take such an iterative approach; rather, they both let the training decide the sparsity pattern. In doing so, Kim et al. [16] relied on an absolute value τ for the threshold. Thus, in their approach, it is difficult to systematically explore the meaningful solutions without problem-specific statistics. Amir and Givargis [17] assigned an integer value to each weight, which is also learned during training, to indicate phase separation. In their approach, due to its autonomous nature, it is not clear how a designer can intervene in the design space exploration.

In the proposed technique, we take the similar iterative approach as Tann et al. [19]. That is, pruning is first applied excessively to obtain the smallest version. If a minimum accuracy threshold is given that the inference results must exceed, then the smallest network that satisfies this requirement can be considered as phase 1. Once this smallest version is determined, from that smallest version on, we restore a set of parameters while keeping the weights belonging to the previous phase *fixed* as depicted in Figure 1. Note that we aim to obtain better accuracy by restoring some weights while keeping the weights from the previous phase remain the same. Thus, both restored and previously existing weights are used in forward pass during training, and weights updates through back propagation only happen to the restored weights.

B. WEIGHT RE-TRAINING

Frankle and Carbin [38] reported the importance of the initial values of the parameters in CNN training; this implies that it is also crucial to decide how to initialize the added weights in the iterative multi-phase CNN training. While it is typical to randomly generate the initial values from a normal distribution, it is difficult to apply this general wisdom in the proposed approach, since a considerable amount of weight values are inherited as fixed values from the previous phase and never change during the re-training. Therefore, as in the lottery ticket hypothesis [38], we independently train a new CNN with the same sparsity and take its their values as initial values of the multi-phase CNN parameters to be re-trained in the corresponding phase.

Note that the ordinary back-propagation methods repeatedly change the value of *all* weights to minimize the loss values at each iteration. On the other hand, recall that in the proposed technique only newly added weights are subject to change; this makes it difficult to maintain the direction of the weight value changes. In order to keep the momentum and direction of the weight changes, we let all the parameters to be modified temporarily by the back propagation, instead of keeping the fixed part unchanged. Then, at the end of each epoch of the training, we overwrite the old values of the weights that belong to the fixed part.

IV. ARCHITECTURE-AWARE SPARSITY

Existing CNN pruning techniques exploit the sparsity at three different levels of granularity: *weight*, *kernel*, and *layer* levels. Although the weight-level pruning such as [14] has the highest degree of freedom in removing parameters, it has been reported that it is difficult to take advantage of this finest granularity pruning in terms of inference speed [16]. Therefore, coarse-grain pruning techniques [16], [19], [33], i.e., kernel- and layer-level, also known as structured sparsity, have been widely used in favor of faster inference.

Pruning granularity affects the system differently from one design concern to another. For example, a finer-grain pruning performs better than a coarse-grain one in terms of accuracy loss while the coarse-grain works better in terms of computation time and memory footprint reduction. Furthermore, in order to enable sophisticated exploration of the design space with respect to the given design

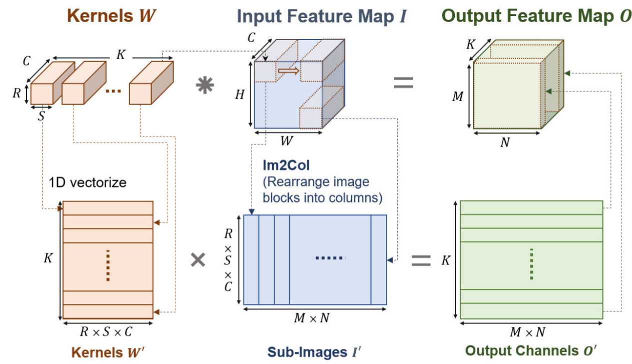


FIGURE 2. GEMM-based computation of a CONV layer: 1) *im2col* is applied to kernels ***W***, input feature map ***I***, and output feature map ***O*** to obtain the converted 2D matrices ***W′***, ***I′***, and, ***O′***. 2) Then, the convolution can be represented as a GEMM operation of ***O′*** = ***W′*** × ***I′***.

concerns, the pruning should be performed in an architecture-aware manner. That is, the granularity of sparsity must be judiciously chosen for a given resource requirement or underlying hardware.

The proposed multi-phase training considers two different levels of granularity in building multi-phase CNNs, i.e., for each phase of multi-phase training, we add and train new parameters

- at the unit of kernel (coarse-grain) or
- at a finer granularity in hardware-specific manner.

While the coarse-grain one is similar to existing kernel-level pruning, the fine-grain one is unique and novel in that the pruning decisions are made considering the underlying hardware architecture. Specifically, we consider the SIMD CPU architecture, since the CPU is the most common commodity hardware for embedded computing. Yu et al. [36] proposed a SIMD-aware pruning technique. However, their approach is not be generally applicable because it is based on a special sparse matrix multiplication format, Compressed Sparse Row (CSR). It has been reported that they only achieved speed up when the sparsity level is above 80%, while the proposed fine-grain pruning is also beneficial for small or moderate sparsity. Layer-level pruning can only be applied to a limited type of CNNs [2] that is equipped with bypass connections. In what follows, we first introduce the general matrix multiply algorithm as a background, which is widely used in CPU-based inference. Then, we present the two different levels of granularity that we use in our technique.

A. GENERAL MATRIX MULTIPLY (GEMM)

Among the layers that make up a CNN, fully connected (FC) and convolutional (CONV) layers are known to be the most computationally intensive. Essentially, an FC layer is the matrix multiplication of the input feature map and the kernel, which is of abundant parallelism. It is also common for the CONV layers to be converted to General Matrix Multiply (GEMM) for efficient inference, as illustrated in Figure 2.

Let us suppose that a CONV layer is characterized as follows:

- H/W: height/width of input feature map,
- C: number of channels in the input feature map,
- R/S: height/width of 3D kernel,
- K: number of 3D kernels in the layer (number of output feature map channels),
- M/N: height/width of output feature map.

Then, the output feature map of this CONV layer $\mathbf{O}$ can be computed as follows:

$$\mathbf{O}[k][n][m] = \sum_{0\leq c<C}\sum_{0\leq i<R}\sum_{0\leq j<S} \mathbf{W}[k][c][i][j] \times \mathbf{I}[c][n+i][m+j], \quad (1)$$

where $0 \leq k < K, 0 \leq n < N$, and $0 \leq m < M$ are the kernel index, the row and column indices of the output feature map $\mathbf{O}$, respectively.

In order to apply GEMM to this convolution operations, the kernel $\mathbf{W}$, which is described as K 3D tensors, is converted into a $K \times (R \cdot S \cdot C)$ 2D matrix $\mathbf{W}'$ as depicted in Figure 2. Similarly, the input and output feature maps $\mathbf{I}$ and $\mathbf{O}$ are also converted to a 2D matrices $\mathbf{I}'$ and $\mathbf{O}'$, which are as big as $(R \cdot S \cdot C) \times (M \cdot N)$ and $K \times (M \cdot N)$, respectively. This procedure is known as *im2col*. Note that this conversion causes a substantial redundancy in the input and output feature maps, i.e., $(R \cdot S \cdot C) \times (M \cdot N) > H \times W \times C$, which is the cost of increased parallelism. With these conversions, the operation of a CONV layer can be simply represented as a single 2D matrix multiplication, $\mathbf{O}' = \mathbf{W}' \times \mathbf{I}'$.

Note that a row vector of the converted matrix $\mathbf{W}'$ is an 1D expansion of all elements of a single kernel that is depicted as a $R \times S \times C$ 3D tensor in the figure. A column vector of the converted input feature map $\mathbf{I}'$ is an 1D expansion of a sub-image. A sub-image corresponds to a subset of the input feature map for a single convolution point. That is, the number of elements in a sub-image is equal to that of a single kernel. So, the total number of sub-images for a CONV layer is determined by how many times the convolution is performed, which is equal to the number of elements in the output channel ($M \times N$). As we have K distinct kernels, the output feature map of the GEMM is represented as $K \times (M \times N)$ as shown in the figure.

It is worthwhile to mention that a column vector of $\mathbf{I}'$ denotes a group of weights that locate at the exactly same position in K different kernels. This will be later exploited in the fine-grain pruning.

B. COARSE-GRAIN KERNEL SPARSITY

Similar to [19] and [39], we consider a structured sparsity for building a multi-phase CNN at a coarse-grain, i.e., a set of weights spanning a certain *contiguous* part of the kernel or a single kernel is considered as a unit of addition in building a multi-phase CNN. The left half of Figure 3 visualizes how kernel-level pruning differs from traditional weight-level

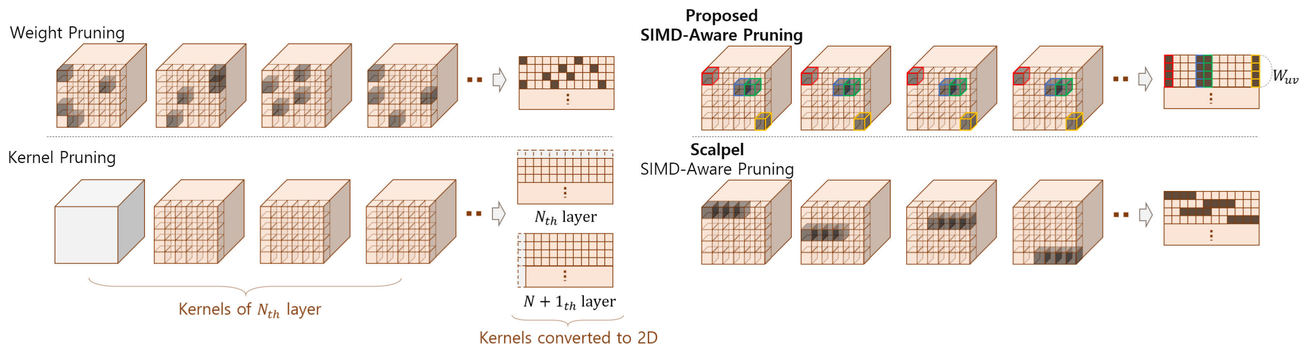


FIGURE 3. Comparison of the pruning approaches: weight-level pruning [14], proposed SIMD-aware (fine-grain) pruning, proposed kernel-level (coarse-grain) pruning, and SIMD-aware pruning of Scalpel [36].

pruning. Traditional weight-level pruning (top left) results in irregularly pruned (zero) values in the converted 2D matrix ($\mathbf{W}'$). On the other hand, kernel-level pruning (bottom left) removes a single kernel (a 3D tensor), resulting in a reduced number of rows in the converted 2D matrix ($\mathbf{W}'$). Because of this, the channel size in the next layer should be adjusted to (as can be seen in the figure). We need to decide which kernels to remove in the kernel-level pruning. As in the traditional pruning method, the l2-norm value of each kernel's elements is used as an indicator of its importance.

It is worth mentioning that the this coarse-grain pruning is memory-friendly thanks to its well-structured spatial locality, i.e., the pruned kernels can be easily removed from memory. Weight-level pruning, on the contrary, would result in irregular patterns of *zeroed-out* weights in the transformed matrices, as illustrated in the figure. Furthermore, kernel-level pruning is also more favorable for achieving a gain in inference speed for any type of hardware. Even without any zero-skipping feature, the number of iterations required by the processing element to perform GEMM operations is automatically reduced.

C. FINE-GRAIN SIMD-AWARE SPARSITY

As stated earlier, the sparsity patterns caused by weight-level pruning typically do not match to the parallelism granularity of the underlying hardware architecture. Therefore, we need another fine-grain sparsity that fits well to the underlying micro-architecture. In our work, we target the finest level of parallelism that exists in the modern microprocessor, namely SIMD operations.

Note that, in a SIMD architecture, multiple operands are processed together at the same time by a vector instruction. For instance, in the case of a 128-bit vector operation (4×32-bit floating point data are computed by a vectorized instruction), four multiplication and accumulation operations are actually computed at the same time in GEMM. Thus, individual zero values that are irregularly scattered in the matrix (the top left case in Figure 3) cannot be skipped efficiently. On the contrary, four contiguous zeros in the converted matrix ($\mathbf{W}'$) may allow safe skipping of the vector instruction (top right in Figure 3). In this example, four elements in $\mathbf{W}'$ that are processed simultaneously by a single SIMD instruction coincide with four different elements at the same position of four consecutive 3D tensors (highlighted with the same color in the figure). This shows how the SIMD-aware skipping is more beneficial in terms of inference time.

This SIMD-aware pruning is applied as a fine-grain pruning to the converted 2D matrix $\mathbf{W}'$. Note that W_{uv} is the SIMD width of the underlying microprocessor, which is 4 in this particular example, but could be different in other hardware. For example, if a processor supports a half-precision quantization (16-bit) and 128-bit SIMD operations can handle 8 operands at the same time, we need to apply $W_{uv} = 8$. Here, as well, we use the l2-norm value to quantify the importance of each pruning candidate.

V. BUILDING A MULTI-PHASE CNN

In this section, we present how to build a multi-phase CNN from an original network, which is exemplified with a 5-phase CNN in Figure 4. We start by describing how to obtain the lowest phase, the smallest CNN from the original CNN using the coarse-grain sparsity. Then, we describe how to determine the size of the highest phase based on the given design requirements and how to iteratively add intermediate phases in addition to the smallest CNN by combining the fine-grain and coarse-grain pruning/restoring.

A. DETERMINING PHASES

1) THE LOWEST PHASE

In the proposed multi-phase training, which builds up from the lower phase to the upper phase, the first thing to do is to determine the lowest phase. The lowest phase has the lowest accuracy, while it tends to have the highest inference speed and the smallest memory requirement. In order to always satisfy the imposed accuracy constraint, this smallest version must also satisfy the accuracy constraint. Therefore, we apply coarse-grain pruning to the original CNN several times in a binary-search manner until we find the pruned CNN with the minimum acceptable accuracy.

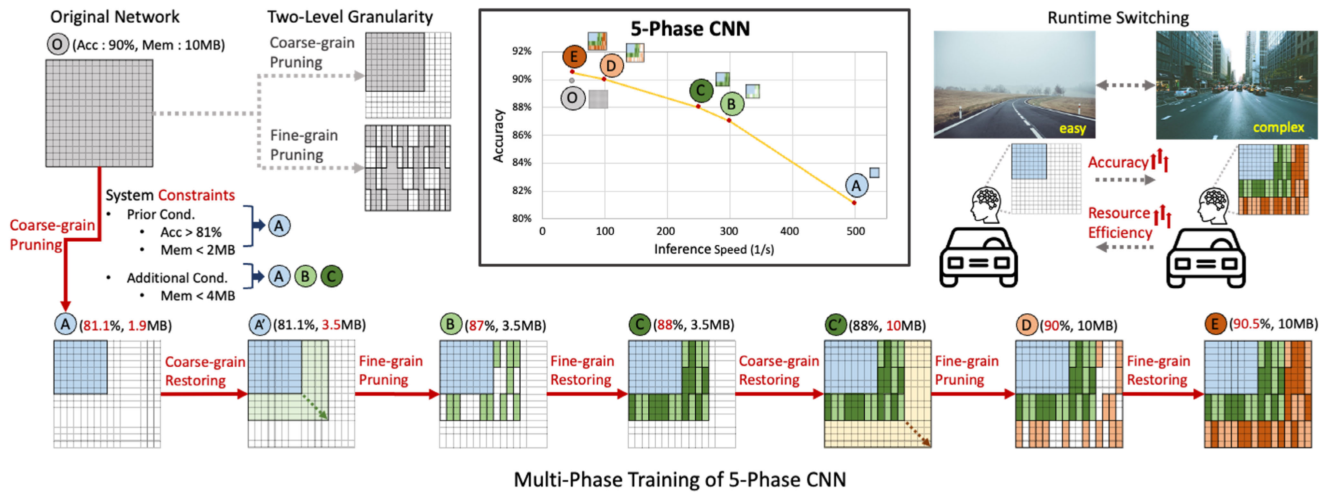


FIGURE 4. How to create a 5-phase CNN using the proposed multi-phase training.

In this case, for instance, we first perform coarse-grain pruning with 50% sparsity and check whether the resultant accuracy is above the minimum acceptable accuracy. If so, we try a more ambitious pruning degree (75%) hoping for obtaining a smaller CNN. If not, coarse-grain pruning is performed again with a smaller sparsity degree of 25%. This procedure is repeated until the resulting accuracy is within the range of the minimum acceptable accuracy $\pm 1\%$. In the proposed technique, we set the minimum acceptable accuracy as 10% less than the accuracy of the original CNN, but this can be set differently depending on the target application. This is exemplified in the left side of Figure 4 in which Ⓐ is derived from the original CNN Ⓞ by applying coarse-grain pruning.

2) THE HIGHEST PHASE

In contrast to the lowest phase, the highest phase uses the greatest amount of resources to achieve the best accuracy and tends to result in the longest inference time. Thus, we choose the highest phase CNN based on the maximum available memory and the minimum inference speed requirement.

While the accuracy of the pruned CNN can be easily quantified using a test or validation dataset, it is not trivial to estimate its inference speed and memory requirement without actually running it. Since it is not feasible to run test inferences during the multi-phase training, we propose to use a polynomial regression-based performance estimation model.

Note that most inference engines running on CPUs, including the one that we use in our implementation [34], are multi-threaded. In such systems, the number of threads is also an important design parameter to optimize. While a larger number of threads can help improve inference speed by increasing parallelism, it can have a negative impact on memory usage due to the duplicated data in GEMM procedures.

We use a third-order polynomial regression analysis to predict the inference speed and memory usage of the system, with inputs of the sparsity of the CNN (i.e., the total number of non-zero parameters) and the number of threads used. Specifically, we created two separate prediction models for inference speed and memory usage. The prediction models were fitted with actual measurements of inference speed and memory usage of various pruned CNNs obtained during the binary search process to find the lowest phase.

Figure 5 shows the result of the prediction model for coarse-grain pruning we used in our experiment. The hyperplane in the figure represents the 3rd order equation obtained from the polynomial regression, and the dots in the figure are the actual measurements for various pruned CNNs. In the experiment, the accuracy of the prediction models for the fitting data was 87.18% and 99.99% for inference speed and memory usage, respectively. Memory usage increases proportionally with the number of threads and network size, making it easier to predict. In contrast, the inference speed demonstrates an irregular behavior; it seems to have a different optimal number of threads for different pruned CNNs. Note that this regression model is target hardware dependent, so a new regression model must be trained if the target hardware changes.

Now that we can estimate the performance of a candidate pruned CNN based on the regression model, we can determine the highest phase CNN; we find the CNN sparsity (the number of non-zero parameters, also referred to as network size) and the number of threads that satisfy the constraints imposed on the minimum inference speed and the maximum memory usage. If there are a number of candidate CNNs that satisfy the given constraints, we select the solution with a larger number of non-zero parameters in the hope of obtaining higher accuracy.

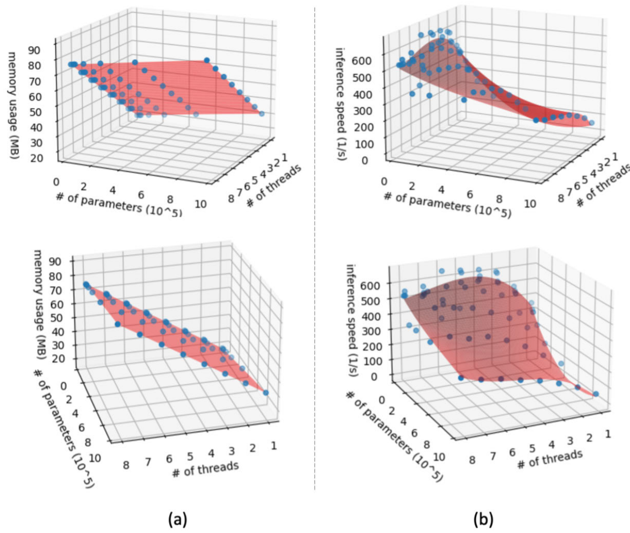


FIGURE 5. Memory usage and inference speed prediction results of CNNs optimized by coarse-grain pruning (red hyperplane: obtained polynomial model, blue dots: actual measurements).

3) INTERMEDIATE PHASES

From the above, we obtained a 2-phase CNN, where the lowest phase is the fastest (but less accurate) and the highest phase is the most accurate given the memory/speed constraints. We can add additional intermediate phases to this. For instance, suppose that we obtained a 2-phase CNN whose parameter sizes are 50MB and 500MB for the lowest and highest phases, respectively. This multi-phase CNN can run on the highest phase to improve accuracy under normal conditions. However, in situations where other applications require more memory space, it could be run in the lowest phase with fewer resources. If it were to run in an intermediate phase that requires about 200MB instead of the lowest phase (50MB), it could maintain a lower level of accuracy degradation while improving memory efficiency.

This increased flexibility comes at the expense of increased training time. In addition, when adding a new phase, the parameters belonging to the lower phases should remain unchanged. This results in a marginal loss of accuracy during training. Adding a larger number of intermediate phases indicates that such negative effects are more likely to be imposed on the multi-phase CNN. Therefore, we need to carefully choose how many phases and how to add them between the lowest and the highest phase.

As mentioned in the previous section, we consider two different sparsity levels: fine-grain and coarse-grain. Using coarse-grained pruning results in a relatively large drop in accuracy, but improves inference speed and effectively reduces memory usage. On the other hand, fine-grained pruning can minimize the drop in accuracy, although the improvement in inference speed and memory usage is relatively weak. Table 1 shows the accuracy results when the two pruning methods are separately applied to obtain a multi-phase CNN, which evidently shows the disadvantage of coarse-grain pruning in terms of accuracy.

TABLE 1. Accuracy comparison of multi-phase CNNs obtained by applying fine-grain and coarse-grain separately.

	Phase	1	2	3	4	5
	Sparsity (%)	94.53	88.82	83.11	41.56	0
Fine-grain	Accuracy (%)	83.86	85.63	86.07	87.63	88.00
Coarse-grain	Accuracy (%)	78.78	81.76	82.89	84.41	86.54

In order to exploit the trade-off between the two pruning methods, we propose to combine them when determining the intermediate phases.[2] We first determine the size of intermediate phase and determine how much to restore at the coarse-grain sparsity. Then, we train the added parameters. This is called coarse-grain restoring and is exemplified in Ⓐ→Ⓐ' in Figure 4. We then perform fine-grain pruning to determine a subset of newly added parameters to survive in the intermediate phase, e.g., Ⓐ'→Ⓑ. Further, we apply a fine-grain restoring to this to obtain another intermediate phase (Ⓑ'→Ⓒ). This fine grain restoring can be repeated until all newly added parameters are trained. The same procedures can be applied between Ⓒ and the highest phase Ⓔ to obtain a 5-phase CNN.

B. RUNTIME SWITCHING

In this subsection, we present how we enable runtime switching between different phases in multi-phase CNN. As stated previously, runtime switching requires adding (or removing) parameters to (or from) an active CNN. In what follows, we show how these are enabled for the two different levels of granularity.

Firstly, for coarse-grain sparsity, we actually load and unload the parameters at runtime. In addition to the parameters that belong to a lower phase, a new set of parameters can be added. We can add more kernels (3-D tensors of W in Figure 2) or/add increase the number of channels (C in Figure 2). This results in an L-shaped additional parameters in the transformed matrix (W' in Figure 2)) as illustrated by Ⓐ→Ⓐ' or Ⓒ→Ⓒ' in Figure 4. It is important to note that the kernels are added or removed in their entirety so that the added or removed kernels only result in a change of number of rows or columns in the transformed matrix W' while keeping its regularity. This allows us load or unload the parameters at runtime.

On the other hand, in fine-grain switching, e.g., Ⓑ→Ⓒ or Ⓓ→Ⓔ, we do not load or unload the parameters at runtime, but choose the ones to participate in the inference by masking. Therefore, added weights for fine-grain sparsity do not need to be stored separately for each phase. Instead, only the masking information needs to be maintained as a meta-data. Consider a sequence of weights $[3, 4, 5]$ as an example in which 3 belongs to the lowest phase (phase 1) and 4 and 5 are additionally included in phases 2 and 3,

[2]Note that, in the proposed technique, the pruning is sometimes applied in an opposite way. That is, as opposed to the traditional pruning methods obtaining an optimized CNN from the largest CNN by *pruning* out some parameters, we start from the smallest CNN and *restore* additional parameters.

TABLE 2. Two benchmark CNNs used in the experiments: VGG-7 trained with CIFAR-10.

Network Configuration
VGG-7
Input
Conv3x3-64
Maxpool
Conv3x3-128
Maxpool
Conv3x3-256
Conv3x3-256
Maxpool
FC-10
Softmax

respectively. In this case, a mask $(1, 0, 0)$ is used for phase 1 while $(1, 0, 1)$ and $(1, 1, 1)$ are used for phases 2 and 3, respectively. In order to minimize the storage requirement, we store this meta-data in a compressed format. That is, we only store the index of the first non-zero weights and the differences of adjacent indices of non-zero weights. For example, for a weight vector $[3, 0, 0, 0, 4, 0, 5]$, the metadata for masking becomes $(0, 4, 2)$, in which the first element 0 in the index of the first non-zero weight and 4 and 2 are the difference between 3 and 4 and 4 and 5, respectively.

VI. EXPERIMENTS

A. EXPERIMENTAL SETUP

To verify the effectiveness of the proposed technique, we performed a set of experiments on top of NVIDIA Jetson AGX Xavier, which is equipped with an 8-core ARM v8.2 64-bit CPU. Although it also includes a GPU, we only utilize the CPU since we target low-power embedded computing. While the multi-phase training framework presented in Section V-A is implemented based on PyTorch [40], the inference engine is separately implemented as an extension of an open source inference engine Darknet with NNPACK [34] as described in Section V-B. Note that NNPACK supports NEON, which is a SIMD-extension of ARM instruction set architecture. We customize the NNPACK NEON library to effectively support the proposed fine-grain pruning which is described in Section IV-C.

In our experimental setup, we quantify the inference speed as the reciprocal of the average inference time measured across all 10,000 images in the CIFAR-10 test dataset. The memory usage is measured by monitoring the memory usage using the top command in Linux while the inference process is on going. This allows us to observe the actual changes in memory usage during inference for different phases of the experiment.

Table 2 summarizes the two CNN models that we used for the experiments: VGG-7. CIFAR-10 were used to train the two models, respectively. The CIFAR dataset is a collection of 32×32 color images, with 10 classes (6,000 images per class), which consist of a total of 50,000 training images and 10,000 test images.

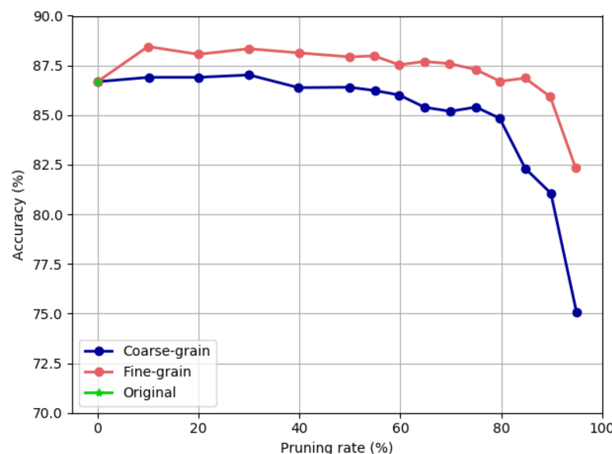


FIGURE 6. Comparison of accuracy according to pruning rate when two different pruning methods are applied to VGG-7 model on CIFAR-10.

B. EFFECTIVENESS OF PRUNING

In the first experiment, we verify the effectiveness of the coarse-grain and fine-grain pruning techniques individually and demonstrate their trade-off in speed and accuracy. We applied each of the two pruning techniques to VGG-7, varying the degree of pruning, i.e., the pruning rate. For instance, a pruning rate of 60% indicates that 60% of the original CNN was removed by pruning.

Figure 6 shows how the accuracy of the pruned CNN evolves over different pruning rates for the two approaches. For both, slight improvements in accuracy have been observed over the original VGG-7 when the pruning rate was lower than 30%. It is known that this anomaly occurs because pruning with a small rate tends to eliminate overfitting of the original CNN. The resulting accuracy of coarse-grain was always lower than that of fine-grain pruning, and the difference became even greater as the pruning rate exceeded 60%. In the case of pruning rates above 85%, we could see a steep drop in accuracy, making it unusable. On the other hand, fine-grain pruning resulted in much better accuracy; it was even higher than that of the original VGG-7 until the pruning rate reached 90%. However, above the 95% pruning rate, there was also a sharp drop in accuracy.

Figure 7 and Figure 8 illustrate the inference speed and memory usage results of the same experiment, respectively. Both pruning approaches showed constant improvement in inference speed as the pruning rate increased. However, the speed gain from coarse-grain pruning was much greater than that from fine-grain pruning. As shown in Figure 8, in the coarse-grain technique, the pruned CNN actually occupied less memory as the pruning rate increased because the weights were actually removed by pruning. On the other hand, fine-grain pruning was less effective in terms of memory usage because the trimmed weight remains in memory with a value of 0.

Figure 6, Figure 7, and Figure 8 evidently demonstrate the trade-offs between the two pruning approaches for accuracy

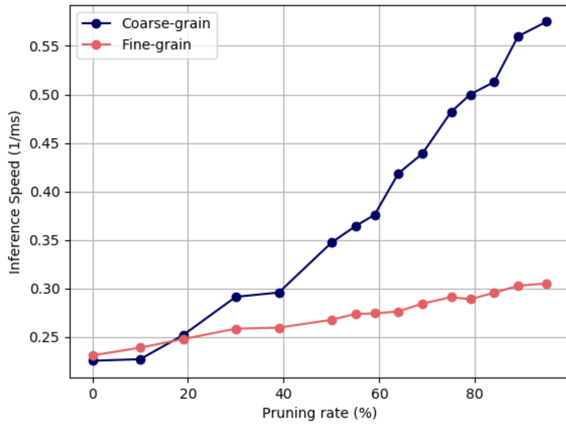


FIGURE 7. Comparison of inference speed according to pruning rate when two different pruning methods are applied to VGG-7 model on CIFAR-10.

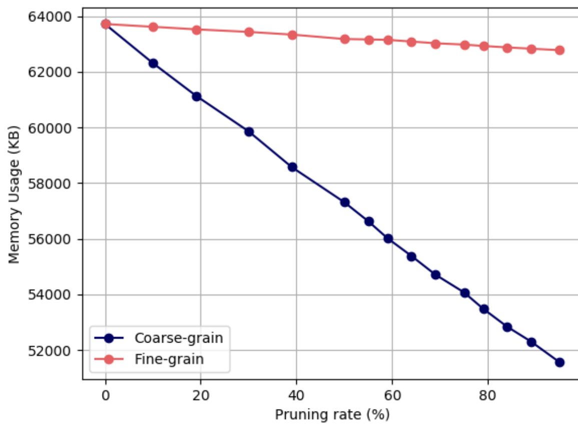


FIGURE 8. Comparison of memory usage according to pruning rate when two different pruning methods are applied to VGG-7 model on CIFAR-10.

and resource usage; fine-grain pruning works better to maintain the accuracy while coarse-grain pruning is preferred to achieve a speedup or reduce resource requirement. In typical use cases, it is common to maximize the accuracy with respect to the given resource budget and latency constraint. We need to carefully balance these two to achieve these two conflicting goals, which is the focus of the next experiment.

C. BUILDING MULTI-PHASE CNN

In this sub-section, we shows how we build a multi-phase CNN from VGG-7 that satisfies a given set of design requirements as a case study.

1) DESIGN CONSTRAINTS/GOALS

We consider the following design constraints:

- **Minimum accuracy constraint**: 10% accuracy loss from the original accuracy,
- **Latency constraint**: an inference should take no longer than 5ms. That is, the inference speed should always be higher than or equal to 0.2 inference per millisecond (ms).
- **Memory budget**: two different memory budgets are considered, 55MB and 65MB.

While satisfying the above design constraints, we want to co-optimize accuracy and inference speed.

TABLE 3. Results of binary search to find most sparse phase.

Binary search step	-	1	2	3
Pruning rate (%)	0	50	75	87.25
Accuracy (%)	86.68	86.36	84.41	83.31
Binary search step	4	5	6	7
Pruning rate (%)	93.75	96.875	95.3125	94.53125
Accuracy (%)	80.25	67.34	76.31	78.78

2) DETERMINING THE LOWEST AND HIGHEST PHASE

As stated in Section V-A1, the lowest phase is determined by the minimum accuracy. The original VGG7 model trained on the CIFAR-10 dataset had accuracy of 86.68%, from which the minimum acceptable accuracy for the lowest phase can be calculated as 78.012%. Coarse-grain pruning was performed repeatedly to find the candidate CNNs for the lowest phase, and the pruning rate was chosen in a binary search manner starting from 50%. Table 3 summarizes the results of the binary search. The first trial with a pruning rate of 50% resulted in an accuracy of 86.36%, which is much higher than the minimum acceptable accuracy. So, in the next step, a higher degree of sparsity, i.e. a pruning rate of 75%, was applied and an accuracy of 84.41% was obtained, which is again above the minimum accuracy constraint. The same procedure was repeated until step 5, where the pruning rate was 96.875%. However, since we obtained an unacceptable accuracy of 67.34% from step 5, a smaller pruning rate was applied in the following step. This binary search ends when the resulting accuracy is within the accuracy margin of 1%p, which is in this particular example between 77.012% and 79.012%.

Note that we obtained a set of differently optimized CNNs with coarse-grain pruning throughout the binary search procedure. These pruned CNNs are used to train the performance prediction models described in Section V-A. Based on these performance prediction models, the highest phase that satisfies the latency constraint (0.2 inference per ms) and the biggest memory budget (65MB). In this particular example, the highest phase was determined to have a pruning rate of 0%, i.e., no pruning is necessary from the original CNN size, with 5 threads in the inference engine.

3) ADDING INTERMEDIATE PHASES

Since we have another smaller memory budget requirement of 55MB (other than the 65MB requirement used to determine the highest phase), additional intermediate phases can be added. Note that these newly added phases are assumed to have the same number of threads as the highest phase for ease of implementation of runtime switching.

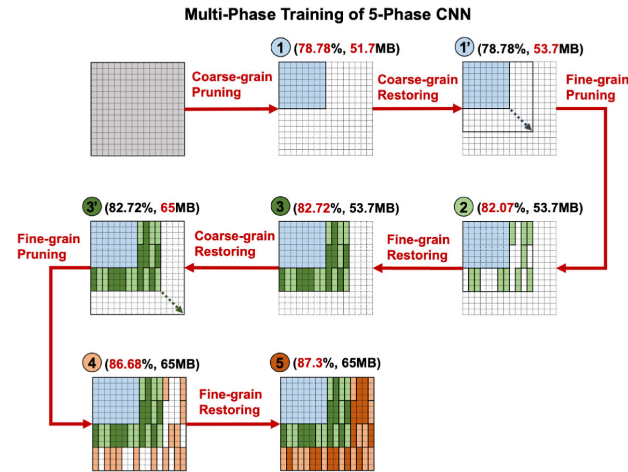


FIGURE 9. Multi-phase training of a 5-phase CNN from a given set of constraints.

TABLE 4. 5-phase CNN configuration that satisfies the four hypothesized conditions.

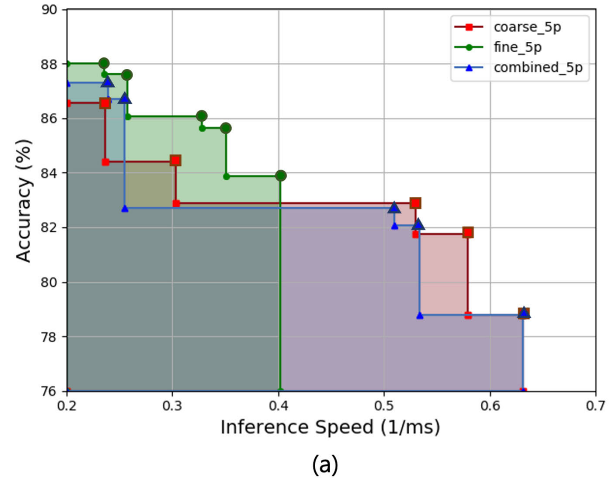
Phase	1	2	3	4	5
Granularity	Coarse	Fine	Coarse	Fine	Coarse
Sparsity (%)	94.53	88.82	83.11	41.56	0

The intermediate phase that satisfies the 55MB memory budget can be defined by means of coarse-grain restoring from the lowest phase. Figure 9 illustrates this procedure; to fully take advantage of the difference between the memory requirement of the lowest phase (51.7MB) and 55MB, a coarse-grain restoring is performed (①→①'). Note that this restored parameters are initialized to be the same weights of the independently trained CNN of the same size (as state in Section III-B). The actual measured memory usage was 53.7 MB, just off the 55 MB estimate. We attribute this error to the prediction error in the regression model and the implementation overhead of the custom inference engine to manage the multi-phase network. Since the regression model's prediction of memory usage is relatively accurate, the overhead is most likely due to differences in the code managing the single network obtained by coarse-grain pruning and the multi-phase network created by adding weights.

Then, a fine-grain pruning is applied to ①' to get an intermediate phase ②. Note that the weights that are present in ①' but not in ② can be additionally trained in the next phase, ③, which will replace ①'. The same procedure is repeated between ③ and the highest phase (⑤). Table 4 summarizes the obtained 5-phase CNN obtained from VGG-7 with respect to the given constraints by the proposed technique.

D. EFFECTIVENESS OF TWO-LEVEL SPARSITY

In this experiment, we show the effectiveness of combining two different levels of sparsity: coarse- and fine-grain sparsity. The multi-phase CNN with 5 phases obtained from



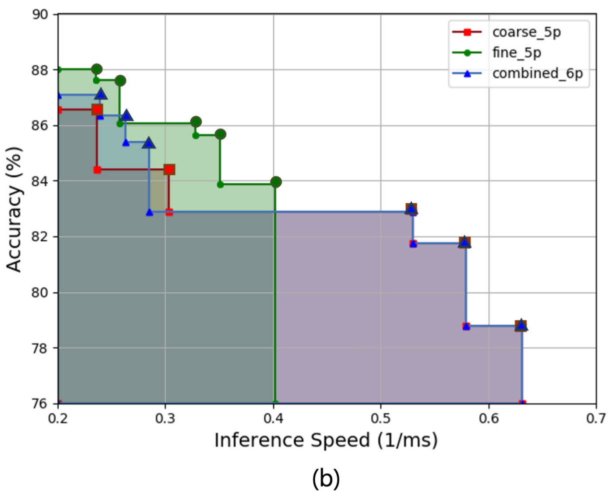


FIGURE 10. Comparison of accuracy and inference speed when multi-phase training is applied to VGG-7 model on CIFAR-10 dataset: (a) proposed 5-phase CNN vs. coarse-grain/fine-grain only and (b) proposed 6-phase CNN vs. coarse-grain/fine-grain only. The colored area denotes the accuracy-inference speed design space covered by the resultant multi-phase CNN.

the proposed combined granularity (*combined_5p*) have been compared with two alternative multi-phase CNNs. The first one (*coarse_5p*) is obtained only from applying coarse-grain pruning/restoring while keeping the sparsity rates as the same as the proposed one. Likewise, the second alternative (*fine_5p*) is obtained from applying only fine-grain pruning/restoring in 5 phases. Figure 10(a) illustrates the result.

While the drop in accuracy was smallest in the case of *fine_5p*, this fine-grain method showed limited ability in terms of inference speed as reported in Table 5. Compared to the proposed multi-phase CNN, it was about 36.4% slower than the proposed approach in its lowest phase. On the other hand, the coarse-grain only method (*coarse_5p*) showed the exactly apposite characteristics; it could be faster in phases 2 and 3 than the proposed one, but it showed worse accuracy results in phases 4 and 5.

TABLE 5. Measurement results of accuracy, inference speed, and memory usage of three 5-phase CNNs and improved 6-phase CNNs.

Phase		1	2	3	(4)	4 (5)	5 (6)
Sparsity		94	88	83	62	41	0
fine_5p	Acc. (%)	83.86	85.63	86.07	-	87.68	88
	Speed (1/ms)	0.4018	0.3509	0.3283	-	0.2573	0.2358
	Mem. (MB)	63.65	63.65	63.65	-	63.65	63.65
coarse_5p	Acc. (%)	78.78	81.76	82.89	-	84.41	86.54
	Speed (1/ms)	0.6317	0.5797	0.5299	-	0.3037	0.2367
	Mem. (MB)	51.71	53.04	54.45	-	60.28	66.61
combined_5p	Acc. (%)	78.78	82.07	82.72	-	86.68	87.3
	Speed (1/ms)	0.6317	0.5336	0.5094	-	0.2546	0.2391
	Mem. (MB)	51.71	53.74	53.74	-	65.02	65.02
combined_6p	Acc. (%)	78.78	81.76	82.89	85.4	86.36	87.08
	Speed (1/ms)	0.6317	0.5797	0.5299	0.285	0.2629	0.2388
	Mem. (MB)	51.71	53.04	54.45	64.39	64.39	64.39

TABLE 6. The hypervolume calculated to quantitatively evaluate the design space of the multi-phase CNNs in Figure 10.

	fine_5p	coarse_5p	combined_5p	combined_6p
hypervolume	2.013187	2.940214	2.739238	3.006089

While the proposed multi-phase CNN is intended to combine the strengths of the two different levels of sparsity granularity, it can be seen that phases 3 and 4 (highlighted in the dashed oval) of the proposed CNN are outperformed by the coarse-grain only method. This can be improved by adding an additional fine-grain phase between these phases, which results in a 6-phase CNN shown as *combined_6p* in Figure 10(b) and Table 5. It can be seen that the proposed technique enables flexible addition of phases to make up the design space.

In order to measure the design space capability of the proposed technique in an objective and quantitative manner, we measure the hypervolume [41] of the design space covered by the resultant multi-phase CNNs. In quantifying the hypervolumes, we used the minimum accuracy and latency constraints reported in Section VI-C1. Intuitively, the hypervolume is a quantitative measure of the colored areas in Figure 10. As summarized in Table 6, the proposed multi-phase CNN outperformed the others in terms of design space exploration capability.

VII. CONCLUSION

In this paper, we propose a multi-phase CNN technique with improved resource efficiency so that CNNs, which have shown high performance in various fields, can be operated in resource constrained systems. In addition, we propose a method for effective design space exploration in the trade-off between accuracy and inference speed of multi-phase CNNs by utilizing two-level granularity. A multi-phase CNN has the same network size as a single CNN while containing sub-networks composed of multiple phases. In addition, by applying two-level granularity to the entire phase, each phase can have a different accuracy and inference speed, allowing for broader and more effective design space exploration. A publicly available CNN inference engine, Darknet with NNPACK, has been adapted to implement the proposed technique, which also allows runtime switching between different phases of multi-phase CNN. A set of design requirements can now be considered together in a single CNN by the proposed multi-phase CNN technique.

ACKNOWLEDGMENT

A preliminary result, Section IV, of this article has been presented in Embedded Systems WEEK 2019, under the title of "A SIMD-aware pruning technique for convolutional neural networks with multi-sparsity levels: work-in-progress [42]." The authors would like to thank Kyusik Choi for his contribution in the implementation of the weight re-training technique (Section III-B).

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[3] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, Feb. 2017.
[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.
[5] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. 18th Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* Munich, Germany: Springer, 2015.
[6] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, and L. Zhang, "NTIRE 2017 challenge on single image super-resolution: Methods and results," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 114–125.
[7] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 136–144.
[8] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck, "GPGPU: General-purpose computation on graphics hardware," in *Proc. ACM/IEEE Conf. Supercomput.*, Nov. 2006, p. 208.
[9] J. Huck, D. Morris, J. Ross, A. Knies, H. Mulder, and R. Zahir, "Introducing the IA-64 architecture," *IEEE Micro*, vol. 20, no. 5, pp. 12–23, 2000.
[10] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
[11] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.
[12] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*.
[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[14] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[15] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[16] E. Kim, C. Ahn, and S. Oh, "NestedNet: Learning nested sparse structures in deep neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8669–8678.

[17] M. Amir and T. Givargis, "Priority neuron: A resource-aware neural network for cyber-physical systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2732–2742, Nov. 2018.

[18] E. Park, D. Kim, S. Kim, Y. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *Proc. 10th Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2015, pp. 124–132.

[19] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *Proc. 11th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2016, pp. 1–10.

[20] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Struct. Multidisciplinary Optim.*, vol. 26, no. 6, pp. 369–395, Apr. 2004.

[21] K. Deb etal., "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proc. 6th Int. Conf. Parallel Problem Solving From Nature*, Paris, France. Berlin, Germany: Springer, 2000.

[22] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," ETH Zurich, Zurich, Switzerland, TIK-Rep. 103, 2001.

[23] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based NoC architectures," in *Proc. 2nd IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Sep. 2004, pp. 182–187.

[24] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 358–374, Jun. 2006.

[25] S. Kang, H. Yang, L. Schor, I. Bacivarov, S. Ha, and L. Thiele, "Multi-objective mapping optimization via problem decomposition for many-core systems," in *Proc. IEEE 10th Symp. Embedded Syst. Real-time Multimedia*, Oct. 2012, pp. 28–37.

[26] N. R. Satish, K. Ravindran, and K. Keutzer, "Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors," in *Proc. 8th ACM Int. Conf. Embedded Softw.*, Oct. 2008, pp. 149–158.

[27] R. Marculescu and P. Bogdan, "Cyberphysical systems: Workload modeling and design optimization," *IEEE Design Test Comput.*, vol. 28, no. 4, pp. 78–87, Jul. 2011.

[28] H. Jayakumar, A. Raha, and V. Raghunathan, "QUICKRECALL: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers," in *Proc. 27th Int. Conf. VLSI Design 13th Int. Conf. Embedded Syst.*, Jan. 2014, pp. 330–335.

[29] X. Li, M. Ye, Y. Liu, and C. Zhu, "Adaptive deep convolutional neural networks for scene-specific object detection," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 9, pp. 2538–2551, Sep. 2019.

[30] K. Tahboub, D. Güera, A. R. Reibman, and E. J. Delp, "Quality-adaptive deep learning for pedestrian detection," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 4187–4191.

[31] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li, "CCR: A concise convolution rule for sparse neural network accelerators," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 189–194.

[32] J. Zhu, J. Jiang, X. Chen, and C.-Y. Tsui, "SparseNN: An energy-efficient neural network accelerator exploiting input and output sparsity," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 241–244.

[33] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[34] J. Redmon, "DarkNet: Open source neural networks in C," 2016. [Online]. Available: http://pjreddie.com/darknet/

[35] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 1–13, 2016.

[36] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN pruning to the underlying hardware parallelism," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, Jun. 2017, pp. 548–560.

[37] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–12. [Online]. Available: https://openreview.net/forum?id=H1gMCsAqY7

[38] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018, *arXiv:1803.03635*.

[39] E. M. Wenzel, "Three-dimensional virtual acoustic displays," in *Proc. Multimedia Interface Design (INCOLL)*, New York, NY, USA, 1992, pp. 257–288. [Online]. Available: http://portal.acm.org/citation.cfm?id=146022.146089

[40] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," *Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[41] D. Brockhoff, J. Bader, L. Thiele, and E. Zitzler, "Directed multiobjective optimization based on the weighted hypervolume indicator," *J. Multi-Criteria Decis. Anal.*, vol. 20, nos. 5–6, pp. 291–317, Sep. 2013.

[42] J. Jang, K. Choi, and H. Yang, "A SIMD-aware pruning technique for convolutional neural networks with multi-sparsity levels: Work-in-progress," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. Companion*, Oct. 2019, pp. 1–2.

JEONGGYU JANG received the B.S. degree in electrical and computer engineering from Ajou University, Suwon, South Korea, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Artificial Intelligence Convergence Network. His current research interests include neural network optimization and acceleration for resource-constrained systems.

HOESEOK YANG (Member, IEEE) received the B.S. degree in computer science and engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2003 and 2010, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Santa Clara University, USA. Before that, he was with Ajou University, South Korea, as an Assistant Professor/Associate Professor, from 2014 to 2021. He was a Postdoctoral Researcher with the D-ITET, ETH Zürich, Zürich, Switzerland, from 2010 to 2014. His current research interests include HW/SW co-design, reliability- and temperature-aware optimization/analysis of multiprocessor system-on-chip (MPSoC), embedded systems design with non-volatile memories, and deep learning for embedded systems.

Dr. Yang has been a Technical Program Committee Member of several conferences or workshops, including the Asia and South Pacific Design Automation Conference (ASP-DAC), the Asia Pacific Conference on Circuits and Systems (APCCAS), the International Conference on Embedded and Ubiquitous Computing (EUC), the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), the Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia), and the International Workshop on Rapid System Prototyping (RSP). Since 2019, he has been serving as an Organizing Committee Member for the Embedded Systems Week (ESWEEK). He was a recipient of the Best Paper Award from the International Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES), in 2012.