Complexity-Driven Model Compression for Resource-Constrained Deep Learning on Edge

Muhammad Zawish, Graduate Student Member, IEEE, Steven Davy, Member, IEEE and Lizy Abraham, Member, IEEE

Abstract—Recent advances in artificial intelligence (AI) on the Internet of Things (IoT) devices have realized edge AI in several applications by enabling low latency and energy efficiency. However, deploying state-of-the-art convolutional neural networks (CNNs) such as VGG-16 and ResNets on resource-constrained edge devices is practically infeasible due to their extensive parameter counts and floating-point operations (FLOPs). Thus, the concept of network pruning as a type of model compression is gaining attention for accelerating CNNs on low-power devices. State-of-the-art pruning approaches, either structured or unstructured, postulate a three-stage: training–pruning–retraining pipeline, which results in an inevitable retraining overhead. In this work, we posit an orthogonal conjecture on structured pruning at initialization to find sparse subnetworks realizing \( \approx 70\% \) parameters, \( \approx 50\% \) FLOPs, and \( \approx 50\% \) memory from CNN layers to guide the filter pruning process without requiring dense pretraining of models. Particularly, we characterize the importance of CNN layers with respect to parameters, FLOPs, and memory-based complexities to work in tandem with filter pruning in a structured manner. Experiments show the competitive performance of our approach in terms of resource acceleration and acceleration for all three modes of pruning, namely parameter-aware (PA), FLOPs-aware (FA), and memory-aware (MA). For example, reducing \( \approx 70\% \) parameters, \( \approx 50\% \) FLOPs, and \( \approx 50\% \) memory from MobileNetV2 did not result in any accuracy loss, unlike state-of-the-art approaches. Lastly, we present a tradeoff between different resources and accuracy, which can be helpful for developers in making the right decisions in resource-constrained IoT environments.

Impact Statement—Deep learning has achieved state-of-the-art accuracy for an array of computer vision tasks. However, enabling CNNs on edge—edge AI—poses significant challenges in terms of resource constraints. Motivated by the idea that CNNs are naturally overparameterized, model compression, particularly pruning, has emerged to enable edge AI by striking a tradeoff between the accuracy and complexity (parameters, FLOPs, and memory) of CNNs. Existing pruning approaches, whether structured or unstructured, assume a three-stage pipeline of training, pruning, and retraining, which leads to retraining overhead. In this study, we show how structured pruning at initialization can find sparse (pruned) subnetworks that require \( 60\% \) less training time than conventional pruning on graphics processing unit (GPU) and, on average, \( 10\text{--}20\% \) less training time on NVIDIA Jetson Nano.

Index Terms—Convolutional neural networks (CNNs), edge AI, edge computing, network compression.

I. INTRODUCTION

D EEP learning as a subbranch of artificial intelligence (AI) has shown remarkable performance in various computer vision applications using extraordinary capabilities of convolutional neural networks (CNNs) [1], [2], [3]. CNNs deployed at Internet of Things (IoT) edges can enable pervasive smart services characterized by exceptional efficiency and strict privacy standards [4].

However, CNNs cannot be ported on resource-constrained edge devices because of their highly parameterized nature. For example, the visual geometry group (VGG)-19 [2] model occupies 500 MBs, and it has a computational overhead of about \( 4E + 08 \) floating-point operations (FLOPs). On the other hand, in several IoT applications, edge nodes are equipped with tiny microcontroller units (MCUs) typically featuring a few megabytes (MBs) of FLASH memory (1–2 MB), several hundred kilobytes (KBs) of random access memory (RAM) (\( \leq 512 \) KB), and single-core CPUs operating at clock speeds ranging between 100 and 400 MHz [5]. As a result, CNNs need to be compressed to reduce resource demands for inference on edge.

Nevertheless, it is increasingly necessary to also address the issue of training on edge due to the following factors: 1) privacy and security of training data are important concerns for many applications (e.g., electronic health records and financial transactions) [6]; 2) as AI-driven products become more personalized, vendors will need to adopt incremental learning capabilities into the edge [7]; 3) the need to reduce the dependency on communication networks and cloud infrastructure to increase global accessibility of AI applications [8]; and 4) several such applications, such as real-time image analysis in autonomous vehicles and interactive virtual reality experiences, demand low latency, which can only be attained by executing computations on edge.
directly on the edge platform [4], [8]. The current edge platforms, however, do not support large-scale CNN training, and this research aims to resolve this caveat.

Over the past decade, CNN compression research has intensified, aiming to reduce CNN memory requirements and computation burdens. Neural network pruning [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] is a type of model compression technique that aims to effectively optimize the CNNs by eliminating redundant neurons and connections subject to the performance of a model’s loss function. Pruning aims to achieve an optimal balance between accuracy and model size in a CNN model. This is achieved primarily by reducing the weight count in a CNN by enforcing zero values for certain unimportant weights. Therefore, performing multiplications and additions with zero is functionally redundant, as multiplying any value with zero results in zero, and adding zero to any number yields the same number. Consequently, there is no need to store or compute with zero weights, and they can be safely omitted to accelerate training/inference.

State-of-the-art pruning approaches have primarily focused on reducing inference costs, leading to the prevalent practice of performing pruning after training. In particular, these approaches follow a three-stage—“train, prune, and retrain” pipeline, resulting in several rounds of training, which makes these experiments computationally inefficient. Therefore, researchers are looking at pruning CNNs early in training, or even before training as training costs, computational demands, and environmental demands have exploded [19]. Lottery ticket pruning (LTP) has meanwhile emerged to overcome conventional iterative retraining. Authors [20], [21] conjectured that pruning CNNs in the early phase of training can lead to compressed subnetworks, which can be then trained in isolation to match the accuracy of the uncompressed (dense) model. These subnetworks are, therefore, termed as lottery tickets. While LTP has sought great attention in pruning literature [22], [23], [24], it does not yet attain a fully efficient training pipeline. These approaches claim to find “lottery tickets” by using insights from training data, but they still need expensive pretraining of a dense model for iterative pruning, which is a notable drawback. We aim to push these boundaries of LTP in this work, to find these lottery tickets (trainable subnetworks) at initialization without requiring the costly pretraining of a dense network. We show that without even warm-up training over a few epochs, unlike typical LTP, our approach is able to find compact CNNs from overparameterized CNN models with competitive performance to state-of-the-art.

Furthermore, traditional research on LTP typically uses unstructured (weight) pruning to get high sparsity while aiming for decent performance in theory [9], [10], [11], [22], [23], [24], [25]. However, unstructured pruning of CNNs is not implementation-friendly as it leads to irregular sparsity, which merely fulfills the expected speedup and throughput in real-world applications. In contrast, many works prior to LTP focused on filter pruning, which seeks to reduce the network by removing unimportant filters [12], [13], [14]. Therefore, in this work, we take a step further to induce structured (filter) pruning at initialization with an aim to provide models with adequately decreased convolution filters besides realistic performance improvements. While state-of-the-art LTP methods such as [22], [23], [24] claim to be successful, they inadvertently complicate pruning regimes through reliance on saliency metrics. For example, authors [20], [21] remove unimportant weights through iterative magnitude pruning (IMP). However, the performance of IMP was questioned in subsequent studies because it introduces computational overhead and also results in loss of performance on complex datasets (e.g., Canadian Institute for Advanced Research (CIFAR)-10 and CIFAR-100) and deeper CNNs (e.g., VGG and ResNets).

In this article, we address the above problems by proposing structured pruning at initialization, which not only accelerates CNN inference but also improves training efficiency on edge. In particular, unlike typical ranking-based approaches, we skip the pretraining and filter-ranking steps due to their unnecessary computational overhead and instead adopt random pruning of filters, which can also achieve comparable or even better performance. We noticed that initializing and training a randomly pruned network from scratch is particularly attractive due to its potential for end-to-end savings throughout the entire training process, in addition to inference benefits. Generally, in most pruning literature [25], [26], [27], random pruning is defined as removing the same proportion of parameters across layers, resulting in uniform layerwise sparsities. These layerwise sparsities fall under random pruning at initialization since they are predefined, and no pretraining is required to determine them.

However, regardless of whether layers are selected for filter pruning uniformly or on an ad-hoc basis, they eventually lead to layer collapse, resulting in a substantial performance loss. This is primarily because previous approaches evenly select candidate layers for pruning and do not consider the heterogeneous underlying nature of complexities being exhibited by convolutional and fully connected layers. As shown in Fig. 1, each layer of VGG-16 architecture contributes differently to each type of complexity, i.e., parameters, memory, and FLOPs. Most of the parameters belong to fully connected layers, while convolutional layers possess most of the FLOPs and also occupy most of the memory size. Hence, it is important to take care of the required level and nature of complexity when pruning models for specialized intelligence for resource-constrained platforms.
Therefore, in this work, we propose complexity-driven pruning, a novel approach that leverages the aforementioned intrinsic complexities of CNN layers to select candidate layers for filter pruning at initialization without requiring dense pretraining of CNNs. Unlike prior methods, where layerwise sparsities are static or determined during training, our approach utilizes the differential nature of the model’s architectural components (i.e., layers) to dictate which layer to select dynamically during the filter pruning process at initialization. In particular, the proposed approach takes into account the distinctive nature and proportion of complexity within each layer and automatically selects the layer for filter pruning based on its contribution to the overall model’s complexity. In this way, we noticed that our approach was able to identify appropriate layerwise sparsities, which turned out to be an important factor toward the successful training of a randomly pruned network from scratch. Moreover, our approach is subtle in a way that it gives a free hand to developers to prune models in three modes, namely parameter-aware (PA), FLOPs-aware (FA), and memory-aware (MA) to achieve different levels of resource and budget requirements. We discuss these modes in detail in Section III. In the subsequent section, we present the novelty and core contributions of the proposed work and the organization of the article.

A. Novelty, Contributions, and Organization of the Proposed Work

In this article, we propose a solution based on complexity-driven compression of CNNs in order to realize IoT-enabled edge AI based applications. We summarize the key contributions of this work as follows:

1) We propose a novel and computationally efficient framework for structured pruning where the pruned model is trained only once (i.e., from scratch), avoiding the computationally intensive filter important and pretraining steps. The target model is first achieved using a proposed complexity-driven pruning algorithm and then trained for a given task. The proposed scheme is illustrated in Fig. 2(b).

2) While pruning filters, we consider the importance/weight of layer-level complexities/characteristics in the overall network to select a particular layer in each iteration. This justifies the overall pruning strategy as layers with more complexity will be pruned more as compared to layers exhibiting less complexity.

3) Effectively, our approach provides versatility by providing different modes of compression such as parameter-based, FLOPs-based, and memory-based. We show that using our method how accuracy can be traded off with varying types of complexities. It can be helpful for developers in repurposing the compression strategy based on the availability of resources.

4) Moreover, we show practical speedups by evaluating the training efficiency of our proposed approach compared to the typical training–pruning–retraining pipeline. The results reflect the significant amount of reduction in training time on resource-rich GPUs and resource-constrained edge devices.

Fig. 2. Comparison between a typical structured pruning pipeline and the proposed complexity-driven approach. (a) Shows a three-stage approach involving computationally intensive ranking and fine-tuning steps. (b) Shows a proposed complexity-driven approach skipping ranking and fine-tuning steps.
5) Finally, we evaluate our proposed approach on AlexNet [1], VGG-16 [2], ResNet-50 [3], and MobileNetV2 [28] using CIFAR-10 and CIFAR-100 datasets [29]. Our results are consistent and essentially provide competitive performance with lesser resource requirements than the state-of-the-art ranking-based approaches.

The rest of the article is organized as follows. The related background on CNN compression, particularly the existing literature on pruning methods, is reviewed in Section II. Section III discusses the motivation and problem design of the proposed work, followed by experimental setup, performance evaluation, and use cases in Section IV. Finally, Section V concludes the proposed work.

II. RELATED WORK

The practical requirements of CNNs have pushed the interests of researchers toward CNN compression where the objective is to either 1) design lightweight and efficient networks [30, 31, 32]; or 2) prune weights/filters from existing CNNs [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 33, 34, 35, 36]; or 3) replace memory-intensive weights by quantized or binary weights [10, 37]; and 4) or train a smaller network using a large (teacher) network as a guide [38]. However, subject to the scope of the article, below we review the state-of-the-art pruning approaches.

A. (Unstructured) Weight Pruning

Pruning weights is a type of unstructured pruning that reduces weight counts of a neural network. Han et al. [9] were the first to use it, removing irrelevant weights/neurons to compress the model size. The authors used the ImageNet dataset and top-1 and top-3 error metrics to train and evaluate the models, claiming compression of up to 90% for the AlexNet and 91% for the VGG-16. Weight sharing based on unimportant connections was performed by [10], and Huffman encoding was used to decrease the weight in to increase the compression rate. Both methods show high compression, but at the cost of high accuracy drop. Authors [39] proposed a pruning approach in which unimportant weights are located, and it was demonstrated that loss changes decrease when weights are compressed. Merging similar neurons (MSNs) [33] prunes the models in a single-shot manner by first clustering the similar neurons and then merging them to produce sparse structures. While authors [40] propose an iterative approach using generative adversarial learning to learn the sparse soft mask, which forces the output of specific structures to be zero. However, all these approaches provide a model with high sparsity, which results in the increased complexity of hyperparameter optimization. Although unstructured pruning methods can reduce memory requirements, it is difficult in general to improve performance due to irregular sparsity in the model. Due to this irregularity, several dense accelerators like GPUs and tensor processing units (TPUs) have a hard time leveraging it effectively. Therefore, it is often found that unstructured pruning often leads to similar or worse performance than dense counterparts due to the extra complexity of compressing and decompressing the weights [41].

B. (Structured) Filter Pruning

Structured pruning, in contrast to the above, has become increasingly popular by overcoming the loopholes of irregular sparsity-induced approaches. The core idea is to achieve structured sparsity by eliminating the parts of the structure, such as channels/filters [12, 13, 14, 35, 36] or blocks [41, 42]. In particular, filter pruning has gained the most traction due to its hardware-friendly nature. What makes filter pruning straightforward is that pruned filters lead to a sparse pattern that can be easily accelerated on commodity hardware platforms, such as NVIDIA Jeston Nano. There is, however, a challenge in determining which filters can be safely pruned without compromising performance. For instance, [12] proposed the use of $l_1$-norm to rank the importance of filters so that the filters with lower $l_1$-norm can be removed as their contribution is insignificant. Alternatively, authors [13] replaced $l_1$-norm with entropy as a measure to calculate filter importance. In this case, the higher entropy of a certain filter reflects its significance in terms of increasing accuracy. In [14], authors proposed the use of partial least squares to capture the relationship of filter importance with class importance in a low dimensional space. Although [12, 13, 14] attribute filter-ranking as novelty, it doesn’t actually contribute in saving accuracy. Unlike traditional large-ratio pruning, [14] uses an iterative pruning approach with small ratios. A similar method is proposed in [35], which uses a rising energy model to identify inactive convolutional kernels. Similarly, [36] prunes layers based on their impact on accuracy. Conventional pruning methods, by contrast, rely primarily on heuristic criteria to identify and prune nonimportant filters and uses typical top-5 accuracy metric for evaluation. However, the key caveat lies in the iterative training–prune–retrain process, potentially leading to an increased overall training cost.

C. Lottery Ticket Pruning

The approaches discussed above follow a three-stage pipeline: training a CNN, pruning based on some ranking criteria, and fine-tuning to regain accuracy. However, authors [20, 21, 43] proposed a lottery ticket theory that unveils that large CNNs consist of sub-CNNs (lottery tickets), which can be trained subsequently in isolation to match the performance of dense networks. To find these lottery tickets, authors employ the following steps: after training a dense network over some epochs, a proportion of weights are pruned based on a saliency metric (akin to aforementioned methods [12, 13, 14, 35, 36]). The remaining weights shape these lottery tickets, which are retrained until convergence. Although these lottery tickets have high training efficiency, their creation incurs additional computational costs since they still require retraining and weight-saliency computations citefrankle2020pruning.

In this work, we address this problem by proposing the idea of training only once, where directly training a pruned model can achieve competitive performance as other approaches. Furthermore, LTP approaches so far focused on fine-grained weight pruning [21, 22, 23, 24], leading to irregular sparse tensor computations, making it hard to accelerate on commercially available deep learning (DL), GPUs, and TPUs. Motivated by
this, we resort to a coarse-grained filter pruning approach, which induces structural sparsity patterns and provides practical energy and latency savings. In terms of filter-saliency metric, it has been reported that despite the importance score, most criteria are incapable of measuring importance at initialization because the weights change drastically after training [44]. Thus, these saliency criteria merely contribute toward performance, instead introducing additional complexity in the pruning pipeline. Alternatively, [45] asserted achieving comparable performance by avoiding filter-ranking altogether and relying on random pruning instead. It has been observed in [45] that CNNs are plastic and can recover from damage during the fine-tuning phase, regardless of which filters are pruned. While the work in [45] still follows conventional training-prune-retrain pipeline, we show in our work the effectiveness of random pruning at initialization. Similarly, none of the above studies consider the underlying complexity proportion of a CNN layer, and they either prune the model layer by layer or all layers at once in a single iteration. This does not justify the pruning strategy, as each layer possesses a different level and nature of complexity, as shown in Fig. 1. Therefore, our work takes inspiration from the natural complexity of CNN layers at initialization and produces a small model (lottery tickets) using complexity-driven pruning. In particular, we construct well-established pruning modes for assigning importance to layers based on their relative weight during filter pruning. The proposed work is able to achieve competitive performance as state-of-the-art techniques on AlexNet [1], VGG-16 [2], ResNet-50 [3], and MobileNetV2 [28] using CIFAR-10 and CIFAR-100 datasets [29].

III. PROPOSED APPROACH

In this section, we discuss the background of our approach, followed by the details on proposed complexity-driven pruning, and a tutorial detail of algorithm.

A. Background

A typical CNN consists of an arbitrary number of convolutional and fully connected layers stacked up on top of each other which account for the overall CNN complexity and performance. A typical convolutional layer \( k = \text{Conv}(X, W) \) receives an input tensor \( X \in \mathbb{R}^{C_{in} \times W \times H} \) and applies a trainable tensor of weights \( W \in \mathbb{R}^{C_{out} \times C_{in} \times K_w \times K_h} \) to produce an output tensor \( O \in \mathbb{R}^{C_{out} \times W \times H} \)

\[
\text{Conv}(X, W)_i = \sum_{c \in [C_{in}]} W_{i,c} \ast X_{c, i} \in [C_{out}] \tag{1}
\]

where \( C_{in} \) and \( C_{out} \) denote the number of input and output channels or filters, respectively. Such convolution operations are responsible for most of the computations inside CNNs during training and inference. Therefore, to enable them for resource-constrained execution, prior works on structured pruning rely on a three-stage pipeline, as shown in Fig. 2(a). At first, a highly parameterized network is trained, then filters are ranked and pruned from one layer or across all layers based on some filter importance criteria, and lastly, the network is fine-tuned on a specific task to minimize the loss. contrast, as shown in Fig. 2(b), we advance the spirit of directly training the compressed model, where the target model is derived using complexity-driven pruning at initialization and then trained only once for a given task. This idea can achieve comparable or even better performance by avoiding the complex ranking and fine-tuning steps that account for a high computational cost in the whole iterative pruning process, as detailed below in Sections III-B and III-C. Firstly, given a predefined CNN topology, our approach achieves the pruned small model by exploiting the intrinsic layer-level complexities, which are in the form of layer parameters, layer-FLOPs, and layer-memory. A particular layer is selected in each pruning iteration for random filter pruning of the predefined percentage of filters. Once the pruned target architecture is achieved, then it is trained on the given dataset. Moreover, our technique helps developers to prune the model based on the available resources. For instance, if the objective is to minimize only the computations while sacrificing the storage, then the pruning mode can be selected as FLOPs and vice versa.

B. Filter Pruning at Initialization

In this subsection, we introduce filter pruning using the convolution layer notations from the previous subsection. For a convolutional layer \( k \), let \( x_{k,1} \in \mathbb{R}^{C_{in} \times W_1 \times H_k} \) be the \( i \)th channel in its input feature map, where \( c_k \) represents the channel count, \( h_k \) represents the input’s height, and \( w_k \) represents the input’s width. Similarly, \( x_{k+1,1} \) denotes the output feature map in the \( k \)th convolution layer and input feature map of the \( k+1 \)th convolution layer. To compute the output feature map of a convolutional layer, denoted as \( x_{k+1,1} \), a filter \( n_{k,1} \in \mathbb{R}^{C_{in} \times \Omega_k \times \Omega_k} \) is applied to the input feature map \( x_k \), following the formulation presented in the following equation. Note that \( \otimes \) denotes the convolution operation and \( \Omega_k \) is the kernel size of convolution filter \( n_{k,1} \).

\[
x_{k+1,1} = n_{k,1} \otimes x_k. \tag{2}
\]

Pruning such filters induces structural sparsity (unlike weight/unstructured pruning) in CNNs, which is crucial for achieving computational efficiency on standard DL frameworks (e.g., TensorFlow and PyTorch). As an example, removing a filter \( n_{k,1} \) will also result in the removal of \( x_{k+1,1} \) channel in the output feature map. Now this output feature map of first layer will serve as input feature map of second layer. Therefore, the removal of this filter will also affect the input feature of the subsequent layer, \( C_{m,1}^{k+1} \). As such, this process will transform its dimension to be \( \mathbb{R}^{(C_{m,1}^{k+1}-1) \times \Omega_{k+1} \times \Omega_{k+1}} \).

Fig. 3 shows the illustration of how does filter pruning works. Blocks in dashed lines are removed by the filter pruning. In this case, since \( n_{k,1} \) has been pruned, it is no longer possible to have \( x_{k+1,1} \). Additionally, it is to be noted that as a result of pruning filter \( n_{k,1} \), only the first input feature \( C_{m,1}^{k+1} \) in the next layer will be removed, while the output feature map will be solely determined by its number of filters after pruning.

1) Pruning From Scratch: Although filter (structured) pruning is renowned for bringing meaningful computational speedups, it is often coupled within a cumbersome three-stage—“train, prune, and retrain” pipeline. Since the goal of state-of-the-art pruning approaches have remained on cutting
down inference cost, mostly pruning is performed after training. In particular, filter pruning techniques that are targeted for inference generally train unpruned networks, prune their unimportant weights, and perform dense retraining (fine-tuning) on pruned network. Following this pruning pipeline, the pretrained weights need to be retrained (using pretrained weights to initialize the new weights) to recover accuracy losses. It should be noted, however, that these pruned networks rarely reach the same level of accuracy when they are retrained from scratch [21]. In this way, conventional filter pruning is not useful for accelerating CNN training because it is targeted at inferencing.

LTP research has meanwhile emerged to pursue efficient training [20], [21], [22], [23], [24], [25]. Inspired from the idea that the computational and storage costs of CNN training are high, LTP aims to generate lottery tickets, which are basically pruned networks when trained from scratch can match the performance of original network. LTP derives these lottery tickets in the following steps:

1) Step 1: Initialize CNN weights \( W \) with random values.
2) Step 2: Start training over \( t \) epochs to obtain trained CNN weights \( W_t \).
3) Step 3: Perform pruning operation based on some predefined saliency/importance criteria and pruning ratio \( P_r \).
4) Step 4: Reinitialize the remaining/unpruned CNN weights \( W_t \) with the original weight values \( W \). After this reinitialization, the CNN with remaining weights is known as lottery tickets.
5) Step 5: Retrain these lottery tickets from scratch over \( t \) epochs to converge on the same dataset.

Repeating the above steps either iteratively or only once can discover lottery tickets, which are claimed to be efficient in both training and inference. However, LTP methods necessitate pretraining a denser model before identifying lottery tickets, which contradicts its original aim of achieving training efficiency. We overcome this caveat by successfully finding lottery tickets (i.e., subnetworks with competitive accuracy) at initialization. In simple words, our approach avoids pretraining and proposes pruning from scratch using the following steps:

1) Step 1: Initialize CNN weights \( W \) with random values.
2) Step 2: Identify lottery tickets using the proposed complexity-driven approach, as discussed in the subsequent section.
3) Step 3: Train these lottery tickets from scratch over \( t \) epochs to converge on a given dataset.

In this case, we avoid pretraining of a dense network and directly train its sparse version from scratch, realizing cost-, and energy-efficient training. Unlike LTP, we show that it is not necessary to train a highly parameterized large model and then prune it; instead, one could directly train the target pruned model to achieve the same level of performance.

2) Filter Importance and Pruning Ratio: Filter importance/saliency defines which filters can be pruned without incurring any performance loss. In that regard, most pruning research focuses on designating novel filter importance metrics without considering the relationship of internal components of a CNN architecture (i.e., layer densities). Generally, these importance/saliency criteria are accustomed to utilizing data from pretraining. For example, IMP assigns scores on weights based on their magnitude after training [25]. However, we argue that if these importance/saliency criteria claim to “only” prune weights that do not contribute toward accuracy, then why does the pruned model do not match the performance of its dense model. Thereby, in our case, we observe that relaxing the pretraining and filter importance requirements and adopting random filter pruning instead can still discover lottery tickets, given that layerwise sparsities are carefully achieved. Moreover, the pruning ratio \( P_r \) indicates how many weights/filters need to be removed from layers in each pruning shot. In general, these ratios are applied uniformly to all layers or decided after training. For example, if \( P_r = 0.2 \), then 20% filters will be removed from all layers in each iteration. However, this mechanism severely disturbs the model architecture and leads to layer collapse in some cases. In our approach, we formulate the decision to apply \( P_r \) on any layer in a data-driven way, utilizing the intrinsic layer-level characteristics in an efficient way without costly pretraining. In the subsequent section, we discuss how to derive these appropriate layerwise sparsities to successfully train lottery tickets from scratch.

C. Complexity-Driven Pruning

Suppose a CNN’s topology consists of \( K \) convolutional layers, and the \( k \)th layer has \( N_k \) filters. For all filters \( N_k \) of \( k \)th layer, we encode the parameters of each filter as \( W_k^m \in \mathbb{R}^{C_k^{in} \times \Omega_k \times \Omega_k} \), where \( C_k^{in} \) denotes number of input channels, and \( \Omega_k \) denotes size of the kernel. Thereby, combining all filters together, we get the filter parameter set \( W = \{ \{ W_k^m \}_{m=1}^{N_k} \}_{k=1}^{K} \), where \( N = \sum_{k=1}^{K} N_k \) denotes the total number of filters in the entire CNN. Then, for a dataset \( D = \{ \{ x_i, y_i \}_{i=1}^{Z} \} \) with \( Z \) inputs and their corresponding labels, the pruning mechanism aims to find a CNN model \( \hat{M} \) with fewer parameters as compared to the baseline CNN model \( M \) using the following equation:

\[
\min_{\hat{W}} L(D|\hat{W}) \quad \text{s.t.} \quad ||\hat{W}||_0 \leq ||W||_0
\]

(3)

where \( L(D|\hat{W}) \) denotes the loss between model predictions and ground truth, and \( \hat{W} \) denotes the remaining parameters in the model \( \hat{M} \).

In each iteration, typical structured pruning focuses on identifying the least important filters and prunes a certain percentage
of filters from either all layers or a single layer. However, this approach is not justified as filters are unevenly distributed across a network’s lower and upper layers, hence resulting in a different number of parameters, FLOPs, and memory requirements. In Fig. 4, we show an example in support of this argument representing the calculation of complexities in a three-layered CNN architecture. It can be seen that inconsistency in the number of filters highly correlates with inconsistency in type and scale of complexity. Thus, it is unfair to prune the same ratio of filters from both a layer with lower complexity and a layer with higher complexity, as it will degrade the performance of the network. To overcome this, we propose the idea of a complexity-driven selection of layers in each pruning iteration using the weighted random sampling technique. Thereby, the likelihood of selection of layer \( k \) in each round is according to the complexity weight associated with it. Let us denote the complexity weight of layer \( k \) be \( \omega_k \), then the probability \( P \) of \( k \) to be randomly sampled is proportional to its relative weight, i.e., \( P_k = \frac{\omega_k}{\sum_{k \in K} \omega_k} \). Note that the complexity weight \( \omega_k \) can be calculated using (4), and FLOPs-driven pruning is formulated using (5) \[
\omega^f_k = C^i_k \times (\Omega^k)^2 \times C^o_k \times S^o_k
\] \[
\min_{k=1}^{K} \sum_{k=1}^{K} \omega^f_k. \tag{5}
\]

2) Parameter-Aware Pruning: In a typical CNN, the filters in convolutional layers account for fewer parameters than the fully connected layers. For instance, in VGG-16, 90% of parameters belong to three fully connected layers, while 10% belong to 13 convolutional layers. Thus, if it is required for a particular application to reduce the parameters, then the proposed technique can be used in the PARAMS mode. In this mode, the parameter space of convolutional layers can be reduced to lower not only the memory footprint but also the operational cost during the inference. To achieve this, for each convolutional layer \( k \), its parameter-based complexity can be calculated using (6), and parameter-driven pruning is formulated using (7) \[
\omega^p_k = C^i_k \times (\Omega^k)^2 \times C^o_k
\] \[
\min_{k=1}^{K} \sum_{k=1}^{K} \omega^p_{k}. \tag{7}
\]

3) Memory-Aware Pruning: The reduction in memory of a CNN is critical when the aim is to achieve both computation and energy efficiency. Since a small model size consists of not only fewer FLOPs but also lesser dynamic random-access memory (DRAM) traffic involving the read and write of feature maps and model parameters. This mode of pruning is often required for compressing CNNs for microcontrollers, as they have a restricted storage budget. Moreover, in order for the deep models to run at the edge, they must fit within the target device’s RAM without disrupting the IoT application at the runtime. To achieve this, the memory-based complexity of each convolutional layer \( k \) can be calculated using (8), and memory-driven pruning is formulated using (9) \[
\omega^m_k = [(\omega^m_k \times 3) + \{(S^o_k \times C^o_k) \times 2\}] \times 4 \tag{8}
\] \[
\min_{k=1}^{K} \sum_{k=1}^{K} \omega^m_k. \tag{9}
\]
Algorithm 1: Complexity-Driven Pruning

1. Input CNN topology $M$ with $K$ convolutional layers
2. Input required complexity $\{C_r\}$
3. Input mode $\{FLOPs, MEMORY, PARAMs\}$
4. Input Pruning ratio $P_r$
5. Begin
   6. switch mode do
      7. case FLOPs do
         8. while $C_M \leq C_r$ do
            9. weights $\leftarrow []$
            10. for $k \in \{1, \ldots, K\}$ do
                11. $\omega_k^f = C_{in}^k \times (\Omega) \times C_{out}^k \times S_{out}^k$
                12. weights.append($\frac{\sum_{i \in K} \omega_i}{\omega_k}$)
            13. end
            14. $k = WRLS(K, weights)$
            15. $k = prune\_filters(k, P_r)$
         16. end
      7. case PARAMs do
         8. while $C_M \leq C_r$ do
            9. weights $\leftarrow []$
            10. for $k \in \{1, \ldots, K\}$ do
                11. $\omega_k^c = C_{in}^k \times (\Omega) \times C_{out}^k$
                12. weights.append($\frac{\sum_{i \in K} \omega_i}{\omega_k}$)
            13. end
            14. $k = WRLS(K, weights)$
            15. $k = prune\_filters(k, P_r)$
         16. end
      7. case MEMORY do
         8. while $C_M \leq C_r$ do
            9. weights $\leftarrow []$
            10. for $k \in \{1, \ldots, K\}$ do
                11. $\omega_k^m = [(\omega_k^f \times 3) + \{(S_{out} \times C_{out}^k) \times 2\}] \times 4$
                12. weights.append($\frac{\sum_{i \in K} \omega_i}{\omega_k}$)
            13. end
            14. $k = WRLS(K, weights)$
            15. $k = prune\_filters(k, P_r)$
         16. end
   8. end
   9. Output the pruned CNN model $M'$ for training

where $K$ is the number of convolutional layers, $k$ denotes the layer index, $C_{in}$ is the number of input channels, $C_{out}$ is the number of output channels, $\Omega$ is the filter size, $S_{out}$ is the feature size of the output layer, and $P_k$ denotes the likelihood of layer $k$ to be selected for pruning. As noted above in (8) and (9), memory consumption is a combination of parameters (i.e., weights and biases) and activations. In the product term on left, the constant value is 3, because we need to consider not only parameters, but also gradients and momentum variables. Similarly, in the product term on right, the constant value is 2 since we also need to store the error of activations during the backpropagation process. Finally, 4 refers to the number of bytes required to store in a 32-bit system.

D. Algorithm Description

This subsection entails the description of pseudocode presented in Algorithm 1. Lines 1–4 defines the input parameters required for the technique, $M$ is the unpruned model, $C_r$ is the required complexity (such as the number of parameters, FLOPs, or memory), mode ensures the objective of layer selection, and $P_r$ is the % of filters need to be pruned from a selected conv layer. We follow a switch–case design scenario in the algorithm to define the pruning strategy for each mode/case, i.e., FLOPs, MEMORY, or PARAMs. For instance, let us consider the case of FLOPs, line 8 starts a while loop which will includes initialization of a list of weights $= []$ to store the relative complexity weight of each layer $k \in \{1, \ldots, K\}$ on line 9 followed by a for loop on line 10. Inside for loop, we iterate through each conv layer, and compute its complexity $\omega_k^f$ using (3), and append its relative weight $\frac{\omega_k^f}{\sum_{i \in K} \omega_i}$ into weights. Then, line 14 sample layer $k$ using weighted random layer selection (WRLS) approach presented in Algorithm 2, and line 15 prunes layer $k$ according to $P_r$ using the prune\_filters method presented in Algorithm 3. The while loop continues until it meets the following condition: $C_M \leq C_r$, meaning that the complexity of CNN
model $M$ must be either less than or equal to the desired complexity $C_r$. Similarly, lines 18–28 applies for MA pruning [using (5)], and lines 29–39 applies for PA pruning (7). Lastly, line 41 produces the pruned output model $M$ for training.

IV. EXPERIMENTS AND RESULTS

This section consists of four subsections: 1) evaluation of VGG-16 and MobileNetV2 on the CIFAR-10 dataset; 2) evaluation of AlexNet and ResNet-50 on the CIFAR-100 dataset; 3) resource-accuracy tradeoff; and 4) the training efficiency of the proposed approach. In Sections IV-A and IV-B, we show a comparison of our approach with state-of-the-art methods in terms of accuracy, FLOPs, memory, and parameters of compressed models. To evaluate the suitability of the proposed approach, Section IV-C highlights the impact of the proposed approach on energy consumption, latency, CPU, and memory utilization.

We implemented the networks using Keras deep learning API with Tensorflow as a backend on the NVIDIA Tesla K20 GPU with 8 GB memory. The OS was Ubuntu 18.04, with Python version 3.6.12, Keras version 2.2.4, and Tensorflow version 1.15. We trained the networks for 150 epochs with a mini-batch size of 128 and a learning rate of $10^{-3}$ using an RMSprop optimizer. We performed all the experiments on five different seeds and averaged the results over them. For resource-constrained benchmarking, we simulated the models on an OpenStack virtual machine (VM) with 2 CPUs, 4 GB of RAM, and 10 GB of the hard disk. To avoid conflicts, we used the same OS and versions of Python, Keras, and Tensorflow. We benchmarked the results on 100 random images and showed the average latency and resource utilization.

A. Evaluation on CIFAR-10

In this subsection, we evaluate the VGG-16 and MobileNetV2 on the CIFAR-10 dataset. There are 60 000 tiny images in the CIFAR-10 dataset categorized into 10 distinct classes with the dimensions of $32 \times 32$. Each class has 6000 images, of which 5000 belong to the training set, while 1000 belong to the test set.

1) VGG-16: Originally, VGG-16 was proposed for the ImageNet dataset, but several works have also reported its substantial performance on CIFAR-10. The architecture of the VGG-16 model is composed of 12 convolutional layers followed by three fully connected layers stacked on top of each other. The last layer represents the number of classes in a particular dataset, e.g., 10 for CIFAR-10, and it possesses softmax as an activation function, while rectified linear unit (ReLU) was used for the rest of the layers. On CIFAR-10, we compare our approach with MSN [33], generative adversarial learning (GAL) [40], and partial least square (PLS) [14]. MSN prunes the models in a single-shot manner by first clustering the similar neurons and then merging them to produce sparse structures. While GAL and PLS are iterative approaches, GAL [40] applied generative adversarial learning to learn the sparse soft mask, which forces the output of specific structures to 0, and PLS [14] applies partial least squares to identify the importance of filters with respect to the class labels.

We pruned VGG-16 architecture in three different modes, i.e., PA, FA, and MA, using the proposed approach described in Section III. It can be observed from Table I that maximum compression, i.e., 91.02%, is achieved with the MA mode, and maximum gain in accuracy, i.e., 1.76%, was observed with PA mode. This compression level is significant regarding CNN acceleration on the microcontrollers where storage is sparse. Since some applications cannot bear the loss of accuracy, then PA mode can be adopted, as it reduced 84.25% parameters while gaining 1.76% of accuracy. In contrast, the single-shot learning-based approach MSN [33] gained only 0.04% accuracy by lowering the approximately same number of parameters as ours. The iterative process by [14] and GAL [40] dropped 9.7% and 3.36% of accuracy, respectively, by reducing the approximately same number of parameters, FLOPs, and memory.

It can be noticed that since IMP-based LTP [25] could induce high compression (due to unstructured nature), but with a 1.4% drop in accuracy, unlike our (PA).

2) MobileNetV2: In MobileNets [28], [30], the standard convolution layer is replaced with a depthwise separable convolutional layer to make them compact as compared to AlexNet or VGG-16. MobileNetV2 [28] is more lightweight than the MobileNetV1 [30], hence, it contains less number of redundant parameters. However, it is still a bottleneck for small datasets.
such as CIFAR-10, as redundancy remains approximately the same. Thus, it is imperative to prune it for acceleration on resource-constrained devices. The architecture of MobileNetV2 is composed of 17 residual blocks with skip connections, followed by a 1 × 1 conv layer, a global average pooling layer, and a softmax layer.

We compare the performance of our approach on MobileNetV2 with measurement of feature extraction ability (FEAM) [17], reduced mobilnet version 2 (RMNv2) [31], width multiplier (WM) [18], DCP [18], NPPM [46], LTP [25], and LTP + ToST [24]. Although LTP and LTP + ToST were able to realize highly compressed models (with 87% and 74% parameters compressed, respectively), but failed to match the accuracy of the dense network. Likewise, FEAM prunes the filters iteratively by ranking their importance in terms of feature extraction ability. Similarly, [18] and [46] are also based on structural pruning. NPPM [46] uses a performance prediction network in the pruning process as a proxy of accuracy, while [18] takes both classification loss and norms into account as part of the pruning process. In contrast, RMNv2 performs architectural modifications by inducing the heterogeneous kernel-based convolutions and mish activations to make the MobileNetV2 even lighter. We show the comparative analysis in Table II. It can be observed that our approach did not lose accuracy at all; instead, it gained 0% in Table II. It can be observed that our approach did not lose accuracy at all; instead, it gained 0% memory in PA, FA, and MA modes, respectively.

Table I

| Approach     | Unpruned (Dense Model) | Postpruning |
|--------------|------------------------|-------------|
|              | Accuracy (%) | FLOPs | Memory (MBs) | Parameters (Millions) | Accuracy (%) | FLOPs (%)↓ | Memory (%)↓ | Parameters (%)↓ |
| MSN [33]     | 93.62 | – | – | – | 0.04 | 38.05 | – | 8.60 |
| GAL [40]     | 93.94 | – | – | – | –3.36 | 45.2 | – | 82.2 |
| PLS [14] (Itr = 1) | 87.05 | 6.65 × 10⁸ | 127 | 33.64 | 0.46 | 28 | 18.38 | 16 |
| PLS [14] (Itr = 5) | 87.05 | 6.65 × 10⁸ | 127 | 33.64 | –1.08 | 67 | 44.51 | 30 |
| PLS [14] (Itr = 10) | 87.05 | 6.65 × 10⁸ | 127 | 33.64 | –9.7 | 88 | 62.9 | 37.62 |
| LTP [25]     | 93.32 | 6.65 × 10⁸ | 127 | 33.64 | –1.4 | – | – | 85 |
| Our (PA)     | 90.96 | 6.61 × 10⁸ | 125 | 33.19 | 1.76 ± 0.03 | 45 ± 0.05 | 49.09 ± 0.03 | 84.25 ± 0.07 |
| Our (FA)     | 90.96 | 6.61 × 10⁸ | 125 | 33.19 | –3.45 ± 0.04 | 89.67 ± 0.08 | 62.19 ± 0.04 | 41.20 ± 0.05 |
| Our (MA)     | 90.96 | 6.61 × 10⁸ | 125 | 33.19 | –8.76 ± 0.07 | 81.13 ± 0.08 | 91.02 ± 0.08 | 49.15 ± 0.07 |

Accuracy shows the gain in accuracy, where a negative sign denotes the loss regarding the original network. ↓ denotes the reduction in percentage w.r.t the unpruned network. Bold entries indicate the individual accuracy change and its relative complexity reduction using our proposed approach.

Table II

| Approach     | Unpruned (Dense Model) | Postpruning |
|--------------|------------------------|-------------|
|              | Accuracy (%) | FLOPs | Memory (MBs) | Parameters (Millions) | Accuracy (%) | FLOPs (%)↓ | Memory (%)↓ | Parameters (%)↓ |
| FEAM [17]    | 74.3 | 1.7 × 10⁹ | – | 5.73 | –12.6 | 25.6 | – | 21.11 |
| RMNv2 [31]   | 94.3 | – | 9.14 | 2.2 | –2.01 | – | 52.7 | 52.2 |
| WM [18]      | 94.47 | – | – | – | –0.30 | 26 | – | – |
| DCP [18]     | 94.47 | – | – | – | 0.22 | 26 | – | – |
| NPPM [46]    | 94.23 | – | – | – | 0.52 | 47 | – | – |
| LTP [25]     | 87.25 | 6 × 10⁶ | 16 | 3.4 | –2.6 | – | – | 87 |
| LTP + ToST [24] | 89.25 | 6 × 10⁶ | 16 | 3.4 | –2.18 | – | – | 74 |
| Our (PA)     | 89.87 | 4.92 × 10⁶ | 17.7 | 1.36 | 0.07 ± 0.01 | 33.05 ± 0.03 | 49.12 ± 0.07 | 70.03 ± 0.08 |
| Our (FA)     | 89.87 | 4.92 × 10⁶ | 17.7 | 1.36 | 0.23 ± 0.03 | 59.77 ± 0.05 | 28.62 ± 0.05 | 18.5 ± 0.02 |
| Our (MA)     | 89.87 | 4.92 × 10⁶ | 17.7 | 1.36 | 0.96 ± 0.07 | 39.48 ± 0.05 | 53.5 ± 0.07 | 30.72 ± 0.06 |

Accuracy shows the gain in accuracy, where the negative sign denotes the loss regarding the original network. ↓ denotes the reduction in percentage w.r.t the unpruned network. Bold entries indicate the individual accuracy change and its relative complexity reduction using our proposed approach.

B. Evaluation on CIFAR-100

In this subsection, we evaluate the AlexNet and ResNet-50 on CIFAR-100 Dataset. CIFAR-100 also has 60,000 images but is categorized into 100 distinct classes. In this case, the distribution of images per class is 600, of which 500 are used as a training set and the remaining 100 as a testing set.
A single residual block can be expressed as follows:

\[ y = F(x, \{ W_i \}) + x \]

In this case, \( x \) represents the input vector, and \( y \) represents the output vector. \( F(x, \{ W_i \}) \) corresponds to the residual mapping function and \( F + x \) corresponds to the shortcut connection and elementwise addition. Although residual blocks do not strongly impact each other, the dimensions of the feature maps in ResNets must be consistent at the beginning and end of each residual unit. For this reason, pruning residual connections is avoided while designing a pruning strategy.

Table IV shows the comparative analysis of our approach with iterative retraining approaches—DualConv [32], SANet [47], SFP [36], and LTP approaches—LTP [25], SFP [22], GraSP [23], and random pruning (ToST) [24]. In particular, Pruned-B proposed a one-shot pruning approach by removing those filters which contribute less toward accuracy, while SANet models their pruning approach as finding the right shift for each feature map in conv layers to induce the sparsity in networks, and PCAS develops an attention-based statistical approach to quantify channel importance and prunes the same number of channels from all conv layers. In contrast, DualConv proposes structural modifications by replacing the conventional 3 × 3 conv operations with stride 1 among all conv layers, apart from the first layer. In Table IV, it can be seen that our approach beats the state-of-the-art mentioned above by reducing 51.33% FLOPs and 30% of memory and gaining 0.34% and 2.23% of accuracy in FA and MA modes, respectively. The worst performance was given in PA mode, where 0.25% dropped in accuracy, still competing with the DualConv and Pruned-B. Regarding LTP approaches—LTP [25], SFP [22], GraSP [23], and random pruning (ToST) [24], it is interesting to observe high compression (90% parametric sparsity) similar to VGG-16 and MobileNetV2 above, but they all leave a huge gap in terms of matching the accuracy of dense—unpruned models.

3) Ablation Study: We performed an ablation experiment to investigate whether the complexity proxies are necessary to reach competitive accuracy with respect to unpruned model. We investigated MobileNetV2 on CIFAR-10 and ResNet50 on CIFAR-100, and present the results in Tables V and VI, respectively. In Table V, the Δ in accuracy of uniform (baseline) and complexity-driven (proposed) layerwise pruning for MobileNetV2 is shown. As a reference, the unpruned (dense) model has an accuracy of 89.87%. In addition, pruning with a uniform layer sampling approach resulted in a loss of 1.35% ± 0.02. This is because all layers are treated equally by the uniform approach, which performs layer sampling blindly.

### TABLE III

| Approach   | Unpruned (Dense Model) | Postpruning |
|------------|------------------------|-------------|
|            | Accuracy (%) | FLOPs | Memory (MBs) | Parameters (Millions) | Accuracy (%) | FLOPs (%) | Memory (%) | Parameters (%) |
| IEM [35]   | 83.31        | 72.86 | 710 × 10^6  | –                  | –             | 4.34        | 40.98     | –           | 35.08          |
| PLS [14] (Itr = 1) | 83.06 | 72.86 | 2.29 × 10^8 | 129                | 33.95         | –           | 13.46     | 4.11        | 2.15           |
| PLS [14] (Itr = 5) | 83.06 | 72.86 | 2.29 × 10^8 | 129                | 33.95         | –           | 45.38     | 17.92       | 8.44           |
| PLS [14] (Itr = 10) | 83.06 | 72.86 | 2.29 × 10^8 | 129                | 33.95         | –           | 62.79     | 40.61       | 18.97          |
| Our (PA)   | 90.28        | –     | 2.29 × 10^8 | 129                | –             | 0.43 ± 0.02 | 20.11 ± 0.05 | 17.66 ± 0.05 | 63.92 ± 0.1   |
| Our (FA)   | 90.28        | –     | 2.29 × 10^8 | 129                | –             | –1.15 ± 0.08 | 79.28 ± 0.09 | 61.59 ± 0.05 | 47.06 ± 0.09   |
| Our (MA)   | 90.28        | –     | 2.29 × 10^8 | 129                | –             | –3.06 ± 0.1  | 71.90 ± 0.08 | 83.31 ± 0.09 | 51.26 ± 0.06   |

Accuracy shows the gain in accuracy, where the negative sign denotes the loss regarding the original network. ▼ denotes the reduction in percentage w.r.t the unpruned network. Bold entries indicate the individual accuracy change and its relative complexity reduction using our proposed approach.

1) AlexNet: Similar to VGG-16, AlexNet is also commonly used as a standard CNN for benchmarking model compression approaches. However, the AlexNet architecture is lighter than VGG-16, as it has only five convolutional layers, which are followed by three fully connected layers. For CIFAR-100, its last layer represents 100 classes having softmax as an activation function, while the remaining layers used the ReLu activation function. For comparison, we report the results of our approach along with ising energy model (IEM) [35], structured filter pruning (SFP) [36], and PLS [14] in Table III. IEM propose an ising energy model to quantify the inactivity of convolutional kernels in order to prune them. In contrast, SFP proposes a relatively simple approach to assign pruning priority to each layer according to its impact on accuracy. It can be observed in Table III that our approach achieves a 63.92% reduction in parameters with 0.43% accuracy gain, 79.28% reduction in FLOPs with only 1.15% loss in accuracy, 93.31% reduction in size with only 3.06% loss in accuracy using PA, FA, and MA modes, respectively. On the other hand, state-of-the-art single-shot approaches [35], [36] could not achieve a similar level of reduction and comparable accuracy loss. Moreover, as opposed to our approach, these methods are not versatile as they only focus on single complexity, which is not feasible in production environments. The iterative approach [14] has also failed to gain performance with significant model reduction as our approach. Nevertheless, these approaches are computationally intensive with complex ranking criteria, which could not contribute to gaining the consistent performance as direct training of pruned model proposed in this work.

2) ResNet-50: ResNet [3] family is designed with skip connections, similar to MobileNets, to provide high accuracy and faster training. ResNet50 is a 50 layers deep architecture with several residual blocks involving two or more conv layers with skip connections to create a path between two residual blocks. A single residual block can be expressed as follows:

\[ y = F(x, \{ W_i \}) + x \]

In this case, \( x \) represents the input vector, and \( y \) represents the output vector. \( F(x, \{ W_i \}) \) corresponds to the residual mapping function and \( F + x \) corresponds to the shortcut connection and elementwise addition. Although residual blocks do not strongly impact each other, the
Table IV

| Approach          | Unpruned (Dense Model) | Postpruning       |
|-------------------|------------------------|-------------------|
|                   | Accuracy (%) | FLOPs | Memory (MBs) | Parameters (Millions) | Accuracy (%) | FLOPs (%) | Memory (%) | Parameters (%) |
| DualConv [32]     | 78.55       | $1.3 \times 10^6$ | – | 34 | – | 0.7 | 29 | – | 26.5 |
| SANet [47]        | 78          | $1.3 \times 10^6$ | – | 16.9 | – | –0.51 | 80 | – | 76 |
| PCAS [15]         | 74.46       | $1.4 \times 10^6$ | – | 17.1 | – | –0.83 | 66.47 | – | 76 |
| Pruned-B [16]     | 74.46       | $1.4 \times 10^6$ | – | 17.1 | – | –1.15 | 56.28 | – | 54.2 |
| SNIP [22]         | 74.91       | – | – | 23.7 | – | –6.5 | – | – | 90 |
| GraSP [23]        | 74.91       | – | – | 23.7 | – | –6.1 | – | – | 90 |
| LTP [25]          | 74.91       | – | – | 23.7 | – | –4 | – | – | 90 |
| Random pruning (ToST) [24] | 74.91 | – | – | 23.7 | – | –9.5 | – | – | 90 |

Our (PA) 79.31 $1.51 \times 10^6$ 90 23.6 $-0.25 \pm 0.05$ 32.21 $\pm 0.08$ 41.59 $\pm 0.1$ 53.4 $\pm 0.08$
Our (FA) 79.31 $1.51 \times 10^6$ 90 23.6 $0.34 \pm 0.07$ $51.33 \pm 0.1$ 42.7 $\pm 0.08$ 35.15 $\pm 0.08$
Our (MA) 79.31 $1.51 \times 10^6$ 90 23.6 $2.23 \pm 0.08$ 19.38 $\pm 0.07$ $30.2 \pm 0.1$ 18.1 $\pm 0.05$

Accuracy shows the gain in accuracy, where the negative sign denotes the loss regarding the original network. ↓ denotes the reduction in percentage w.r.t the unpruned network. Bold entries indicate the individual accuracy change and its relative complexity reduction using our proposed approach.

Table V

| Approach       | Accuracy | Δ in Accuracy (%) |
|----------------|----------|-------------------|
| Unpruned (dense) | 89.87 | – |
| Pruned (uniform) | 88.65 | $-1.35 \pm 0.02$ |
| Pruned (PA)     | 89.93 | $0.07 \pm 0.01$ |
| Pruned (FA)     | 90.07 | $0.23 \pm 0.03$ |
| Pruned (MA)     | 90.73 | $0.96 \pm 0.07$ |

Δ in Accuracy shows the change in accuracy, where the negative sign denotes the loss regarding the original unpruned network. Bold entries indicate the individual accuracy change and its relative complexity reduction using our proposed approach.

Table VI

| Approach       | Accuracy | Δ in Accuracy (%) |
|----------------|----------|-------------------|
| Unpruned (dense) | 79.31 | – |
| Pruned (uniform) | 76.90 | $-3.40 \pm 0.06$ |
| Pruned (PA)     | 79.11 | $-0.25 \pm 0.05$ |
| Pruned (FA)     | 79.57 | $0.34 \pm 0.07$ |
| Pruned (MA)     | 81.07 | $2.23 \pm 0.08$ |

Δ in Accuracy shows the change in accuracy, where the negative sign denotes the loss regarding the original unpruned network. Bold entries indicate the individual accuracy change and its relative complexity reduction using our proposed approach.

contrast, PA, FA, and MA are proposed complexity-driven pruning modes that differentiate layers by parameters, FLOPs, and memory size, respectively. As a result of these distinctive layerwise sparsities in PA, MA, and FA modes, we witness the accuracy improvements of 0.07% ± 0.01, 0.23% ± 0.03, and 0.96% ± 0.07, respectively. Similarly, Table VI shows the ablation of ResNet50, where unpruned (dense) achieved an accuracy of 79.31%. It can be observed that with uniform layerwise pruning, accuracy declined to 76.90%, resulting in a loss of 3.04% ± 0.06. In contrast, the proposed complexity-driven layer selection modes (PA, FA, and MA) exhibited considerable accuracy changes of $-0.25% \pm 0.05, 0.34% \pm 0.07,$ and 2.23% ± 0.08, respectively. It is evident from these results that it is essential to consider the distinct characteristics of each layer during the pruning process rather than treating them uniformly.

C. Resource-Accuracy Tradeoff for Resource Constrained Execution

It is imperative to evaluate the impact of compression strategies on resource utilization. There is limited attention paid to the end-level benefit of pruning approaches, and the focus has only been on compressing a single complexity with theoretically attractive importance criteria. For example, there is no use in reducing the FLOPs only when executing a microcontroller, as it requires a model with lower memory consumption. Thus, it is critical to answering questions such as how can we measure the effectiveness of an approach on lesser resource consumption when deployed in production? How can we make a suitable tradeoff between different complexities and performance for an optimal deployment? Several metrics exist to evaluate the suitability of compressed models, such as response time (latency), energy consumption, CPU utilization, and memory utilization. Figs. 5 and 6 present a comprehensive tradeoff between accuracy and different types of resource consumption achieved using our approach in PA, FA, and MA mode for VGG-16 on CIFAR-10 and AlexNet on CIFAR-100, respectively.

1) Latency: Apart from theoretical speedups, i.e., reduction in FLOPs, we also evaluate the practical speedups, i.e., inference time which is also known as latency. We measured the latency as a delay in output required by models to classify an image. Figs. 5 and 6 show accuracy-latency tradeoffs of VGG-16 and AlexNet, respectively. It is clear that we can achieve minimum latency using our approach in MA mode at the cost of significant accuracy loss. However, in FA mode, we can not
only minimize latency but also maintain the required accuracy. The inconsistency in FA and MA mode is essentially caused by the unpruned fully connected layers, DRAM access, and the nonparametric layers such as ReLu or Pooling.

2) Energy Consumption: The energy consumption of a model during inference is considered a critical metric to measure its efficiency in resource-constrained production environments. Based on [48], neither the FLOPs nor the number of parameters alone reflects the actual energy consumption of CNNs. Thereby, a model’s total energy consumption is based on energy used in data access and energy required for arithmetic operations on devices. For instance, if each 32-bit FLOP needs 2.3 pJ, then energy for arithmetic operations can be calculated as a product of FLOP counts and 2.3 pJ. Similarly, for data access, retrieving 1MB from DRAM requires 640 pJ, then the product of model size and 640 pJ gives us energy consumption of data access for each model. As shown in Figs. 5 and 6, the energy consumption of both VGG-16 and AlexNet is consistent with the reduction of parameters, FLOPs, and model size. Thus, all three modes are critical in determining the energy usage of a model. Our approach is helpful in such cases, mainly where a developer can compress the model from different aspects, unlike other techniques. Moreover, our approach can help in making the suitable tradeoff between energy and accuracy when it comes to low-energy based execution environments such as microcontrollers.

3) CPU and Memory Utilization: Among other key metrics, CPU and memory usage can also be helpful for developers to configure the hardware or cloud infrastructure correctly. When performing a CNN inference, measuring the CPU and RAM usage involves determining the scale of these resources being consumed. The metric for both resources varies from 0% to 100%. Practically, the percentage metric is not interesting information itself, but the duration of the resource being used is valuable. Effectively, such information helps developers in managing multiple tasks in parallel if more processing or RAM capacity is available. To meet the tight Quality of Service (QoS) constraints, a developer can compress a CNN in a particular mode based on the usage of CPU and RAM. Figs. 5 and 6 show the impact of all three modes on both of these resource usages.

D. Training Efficiency

In this subsection, we evaluate the proposed approach in terms of training efficiency on a resource-rich GPU and a
resource-constrained GPU to justify the key idea of reducing the training bottleneck and achieving competitive performance.

1) **Evaluation on Resource-Rich GPU**: In Table VII, we show the comparison of our approach with a conventional iterative pruning approach, i.e., PLS [14]. Most of the conventional approaches are alike since the methodology follows the training–pruning–fine-tuning procedure. The resource-rich GPU used for training is Nvidia Tesla K20. We follow the training parameters originally mentioned in [14] for training the baseline model and fine-tuning the pruned model after 10 iterations. The models are trained for 150 epochs before and after pruning. Hence, in the case of PLS, the baseline (unpruned) AlexNet took 3.46 h, while the retraining/fine-tuning of the pruned model took 1.91 h. Therefore, the total time in achieving the pruning objective required 5.37 h which indicates the overall training cost by compromising the 4.1% of accuracy.

| Approach                  | AlexNet-CIFAR-100 | VGG-16-CIFAR-10 |
|---------------------------|-------------------|-----------------|
|                           | Training time     | Accuracy (%)    | Δ in accuracy (%) | Training time | Accuracy (%) | Δ in accuracy (%) |
| PLS (Unpruned)            | 3.46              | 83.06           | –                | 3.83         | 87.05       | –                |
| PLS (fine-tuning)         | 1.91              | 79.65           | –4.1             | 1.91         | 78.6        | –9.7             |
| PLS (train + fine-tuning) | 5.37              | 79.65           | –4.1             | 5.74         | 78.6        | –9.7             |
| Our (Unpruned)            | 2.86              | 90.28           | –                | 3.79         | 90.96       | –                |
| Our (PA)                  | 2.83 (50.2% ↓)    | 90.66           | 0.43             | 3.37 (53% ↓) | 92.56       | 1.76             |
| Our (FA)                  | 1.99 (58.96% ↓)   | 89.24           | –1.15            | 2.65 (65.5% ↓)| 87.82       | –3.45            |
| Our (MA)                  | 2.71 (51.4% ↓)    | 87.51           | –3.06            | 2.41 (61% ↓) | 82.9        | –8.76            |

(%) shows reduction in % with respect to baseline.
In contrast, our approach reduced the training time by 50.2% in PA mode, 59.96% in FA mode, and 51.4% in MA mode, along with maintaining competitive accuracy. This kind of training efficiency can be achieved with our approach as it relies on training the pruned model directly instead of following the training-pruning-fine-tuning procedure.

2) Evaluation on Resource-Constrained Edge Device: In Table VIII, we show the training performance of our approach on a relatively resource-constrained edge device, i.e., NVIDIA Jetson Nano, which is often utilized for edge AI applications. This edge device comprises a GPU with 128 NVIDIA CUDA® cores, a Quadcore ARM Cortex-A57 CPU, 4 GB of RAM, and 16 GB of storage. Traditionally, due to its resource-constrained nature, this device is used to facilitate deep learning inference using predeployed models trained on a certain application. However, in this article, we consider this device only for benchmarking the training efficiency of our approach. Since the device has very sparse resources, training 150 epochs for each model would require a lot of time. Thus, for the sake of the evaluation, we trained the AlexNet and VGG-16 for only 1 epoch, with a batchsize of 5, on 20% training data of CIFAR-100 and CIFAR-10, respectively. In each pruning mode, we prune the models up to 50 iterations and report the obtained results in Table VIII. It can be seen that, for both AlexNet and VGG-16, FA mode has performed best in terms of reducing training time as well, such as for the purpose of fine-tuning fresh data or contributing to a federated learning task.

V. CONCLUSION

In this article, we present a complexity-driven structured pruning, which enables compression of CNNs in different modes, i.e., PA, FLOPs-aware, and MA. It is obvious from the above discussion that every layer of a CNN not only shows a different nature of complexity but also contributes differently to overall model complexity. Thus, unlike state-of-the-art approaches, our proposed work takes care of both these aspects when pruning a certain model. The proposed pruning scheme is computationally efficient since every model is first pruned and then directly trained instead of typical three-stage pipelines. As shown in the results, the proposed method can accelerate CNNs in different pruning modes without losing much accuracy. Moreover, developers can benefit from this approach by compressing CNNs in different modes to trade accuracy with various resources. In the future, we aim to utilize this approach for distributed execution on heterogeneous edge devices for real-time inference.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.
[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.
[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[4] A. Luš, M. A. Patricio, A. Berlanga, and J. M. Molina, “Seamless transition from machine learning on the cloud to industrial edge devices with Thinger.io,” *IEEE Internet Things J.*, vol. 10, no. 18, pp. 16548–16563, Sep. 2023.
[5] M. Grimaldi, L. Mocerino, A. Cipolletta, and A. Calimera, “Dynamic ConvNets on tiny devices via nested sparsity,” *IEEE Internet Things J.*, vol. 10, no. 6, pp. 5073–5082, Mar. 2023.
[6] X. Liu et al., “Privacy and security issues in deep learning: A survey,” *IEEE Access*, vol. 9, pp. 4566–4593, 2021.
[7] P. Siirtola and J. Röning, “Incremental learning to personalize human activity recognition models: The importance of human AI collaboration,” *Sensors*, vol. 19, no. 23, 2019, Art. no. 5151.
[8] A. Y. Ding et al., “Roadmap for edge AI: A Dagstuhl perspective,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 52, no. 1, pp. 28–33, 2022.
[9] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” 2015, arXiv:1502.01851.
[10] 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.
[11] Y. Guo, A. Yao, and X. Chen, “Dynamic network surgery for efficient DNNs,” 2016, arXiv:1608.04493.
[12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient ConvNets,” 2016, arXiv:1608.08710.
[13] J.-H. Luo and J. Wu, “An entropy-based pruning method for CNN compression,” 2017, arXiv:1705.05791.
[14] A. Jordan, F. Yamada, and W. R. Schwartz, “Deep network compression based on partial least squares,” *Neurocomputing*, vol. 406, pp. 234–243, Jun. 2020.
[15] K. Yamamoto and K. Maeno, “PCAS: Pruning channels with attention statistics for deep network compression,” 2018, arXiv:1806.05382.
[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient ConvNets,” 2016, arXiv:1608.08710.
[17] H. Wu, Y. Tang, and X. Zhang, “A pruning method based on the measurement of feature extraction ability,” *Mach. Vis. Appl.*, vol. 32, no. 1, pp. 1–11, 2021.
[18] Z. Zhuang et al., “Discrimination-aware channel pruning for deep neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.
[19] D. Patterson et al., “Carbon emissions and large neural network training,” 2021, arXiv:2104.10350.
[20] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin, “Linear mode connectivity and the lottery ticket hypothesis,” in Proc. Int. Conf. Mach. Learn., PMLR, 2020, pp. 3259–3269.
[21] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” 2018, arXiv:1803.03635.
[22] N. Lee, T. Ajakanth, and P. H. Torr, “SNIP: Single-shot network pruning based on connection sensitivity,” 2018, arXiv:1810.02340.
[23] C. Wang, G. Zhang, and R. Grosse, “Picking winning tickets before training by preserving gradient flow,” 2020, arXiv:2002.07376.
[24] A. K. Jaiswal, H. Ma, T. Chen, Y. Ding, and Z. Wang, “Training your sparse neural network better with any mask,” in Proc. Int. Conf. Mach. Learn., PMLR, 2022, pp. 9833–9844.
[25] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, “Pruning neural networks at initialization: Why are we missing the mark?” 2020, arXiv:2009.08576.
[26] T. Gale, E. Eisen, and S. Hooker, “The state of sparsity in deep neural networks,” 2019, arXiv:1902.09574.
[27] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, “Pruning neural networks without any data by iteratively conserving synaptic flow,” in Proc. Adv. Neural Inf. Process. Syst., vol. 33, 2020, pp. 6377–6389.
[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 4510–4520.
[29] A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.
[30] A. G. Howard et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861.
[31] M. Ayi and M. El-Sharkawy, “RMNv2: Reduced Mobilenet V2 for CIFAR10,” in Proc. 10th Annu. Comput. Commun. Workshop Conf. (CCWC), Piscataway, NJ, USA: IEEE Press, 2020, pp. 0287–0292.
[32] J. Zhong, J. Chen, and A. Mian, “DualConv: Dual convolutional kernels for lightweight deep neural networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 34, no. 11, pp. 9528–9535, Nov. 2023.
[33] G. Zhong, W. Liu, H. Yao, T. Li, J. Sun, and X. Liu, “Merging similar neurons for deep networks compression,” Cogn. Comput., vol. 12, pp. 1–12, Jan. 2020.
[34] A. Salama, O. Ostapenko, T. Klein, and M. Nabi, “Prune your neurons blindly: Neural network compression through structured class-blind pruning,” in Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP), 2019, pp. 2802–2806.
[35] H. Salehinejad and S. Valae, “Pruning of convolutional neural networks using Ising energy model,” in Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP), Piscataway, NJ, USA: IEEE Press, 2021, pp. 3935–3939.
[36] M. V. Carballo and B. K. Lee, “Accuracy-aware structured filter pruning for deep neural networks,” in Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI), Piscataway, NJ, USA: IEEE Press, 2020, pp. 679–682.
[37] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1,” 2016, arXiv:1602.02830.
[38] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” 2015, arXiv:1503.02531.
[39] M. A. Carreira-Perpinan and Y. Idelbayev, “Learning-compression algorithms for neural net pruning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 8532–8541.
[40] S. Lin et al., “Towards optimal structured CNN pruning via generative adversarial learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 2790–2799.
[41] X. Ma et al., “Non-structured DNN weight pruning—Is it beneficial in any platform?” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 9, pp. 4930–4944, Sep. 2022.
[42] Z. Tan et al., “PCNN: Pattern-based fine-grained regular pruning towards optimizing CNN accelerators,” in Proc. 57th ACM/IEEE Des. Automat. Conf. (DAC), 2020, pp. 1–6.
[43] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” 2018, arXiv:1810.05270.
[44] A. Jordao, G. C. de Araujo, H. d. A. Maia, and H. Pedrini, “When layers play the lottery, all tickets win at initialization,” 2023, arXiv:2301.10835.
[45] D. Mittal, S. Bhardwaj, M. M. Khapra, and B. Ravindran, “Studying the plasticity in deep convolutional neural networks using random pruning,” Mach. Vis. Appl., vol. 30, no. 2, pp. 203–216, 2019.
[46] S. Gao, F. Huang, W. Cai, and H. Huang, “Network pruning via performance maximization,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2021, pp. 9270–9280.
[47] G. B. Hacene, C. Lassance, V. Gripon, M. Courbariaux, and Y. Bengio, “Attention based pruning for shift networks,” in Proc. 25th Int. Conf. Pattern Recognit. (ICPR), Piscataway, NJ, USA: IEEE Press, 2021, pp. 4054–4061.
[48] S. Han et al., “EIE: Efficient inference engine on compressed deep neural network,” ACM SIGARCH Comput. Archit. News, vol. 44, no. 3, pp. 243–254, 2016.