© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The final version of record is available at http://dx.doi.org/10.1109/TCAD.2020.3013050

Fusion-Catalyzed Pruning for Optimizing Deep Learning on Intelligent Edge Devices

Guangli Li, Xiu Ma, Xueying Wang, Lei Liu, Jingling Xue, and Xiaobing Feng

Abstract—The increasing computational cost of deep neural network models limits the applicability of intelligent applications on resource-constrained edge devices. While a number of neural network pruning methods have been proposed to compress the models, prevailing approaches focus only on parametric operators (e.g., convolution), which may miss optimization opportunities. In this paper, we present a novel fusion-catalyzed pruning approach, called FUPRUNER, which simultaneously optimizes the parametric and non-parametric operators for accelerating neural networks. We introduce an aggressive fusion method to equivalently transform a model, which extends the optimization space of pruning and enables non-parametric operators to be pruned in a similar manner as parametric operators, and a dynamic filter pruning method is applied to decrease the computational cost of models while retaining the accuracy requirement. Moreover, FUPRUNER provides configurable optimization options for controlling fusion and pruning, allowing much more flexible performance-accuracy trade-offs to be made. Evaluation with state-of-the-art residual neural networks on five representative intelligent edge platforms, Jetson TX2, Jetson Nano, Edge TPU, NCS, and NCS2, demonstrates the effectiveness of our approach, which can accelerate the inference of models on CIFAR-10 and ImageNet datasets.

Index Terms—Deep learning system, edge intelligence, neural networks, model compression and acceleration.

I. INTRODUCTION

D EEP neural networks (DNNs) have achieved remarkable performance in various intelligence tasks, such as object recognition [1], and become increasingly popular in mobile and embedded platforms [2]. However, the inference of neural networks, i.e., obtaining the predicted result by a pre-trained model with a given input is a computation-intensive task, performed cumbersomely on edge devices, which is limited by the tight resource constraints, including processor, energy and memory. Despite the fact that the inference procedure can be performed in the cloud, i.e., computation of DNNs can be offloaded to high-performance devices (e.g., cloud servers), this approach is not suitable in several scenarios due

This work is supported by the National Key R&D Program of China (2017YFB1003103), the Science Fund for Creative Research Groups of the National Natural Science Foundation of China (61521092), and the Australian Research Council Grants (DP170103956 and DP180104069). (*Corresponding author: Lei Liu.*)

G. Li, X. Wang, L. Liu, and X. Feng are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China, and also with the University of Chinese Academy of Sciences, Beijing 100190, China (email: liguangli@ict.ac.cn, wangxueying@ict.ac.cn, liulei@ict.ac.cn, fxb@ict.ac.cn).

X. Ma is with the College of Computer Science and Technology, Jilin University, Changchun 130012, China (email: maxiu18@mails.jlu.edu.cn).

J. Xue is with the School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia (email: jin-gling@cse.unsw.edu.au).



Fig. 1. Comparing optimized neural network operators in existing pruning methods with FUPRUNER. The operator to be optimized is marked as the yellow box while the pruned operator is marked as the green box. The pruned operator reduces the number of parameters compared with the original one and is marked with an "*". Especially, the element-wise addition operator has been removed after pruning in FUPRUNER, as marked by the dashed gray box.

to the security concerns, real-time constraints, and unstable network connectivity. As such, the technique for supporting on-device inference has sparked an interest in both industry and academia [3], [4]. On the one hand, several optimization methods for compressing and accelerating neural networks, such as pruning [5], have been proposed, which bring tolerable accuracy loss but decrease the cost of computation and storage. On the other hand, developing intelligent edge devices equipped with specialized hardware, such as Google Tensor Processing Unit (TPU) [6] and Intel Neural Computer Stick (NCS) [7], becomes an inevitable trend due to the inefficiency of general-purpose hardware platforms (e.g., CPU) for executing intelligent applications.

Neural network pruning is one of the effective ways to optimize models by reducing redundant neurons and connections. Prior works on weight pruning [8]–[10] have succeeded in achieving high sparsity as well as theoretically attractive speedups. However, the non-structured schemes (i.e., irregular sparsity) of pruned DNN operators are not implementation friendly, which can hardly reach expected acceleration in realworld applications. In contrast, filter pruning methods [11]-[13], which selectively reduce unimportant filters, shrink a model into a thinner one and achieve structured sparsity. As such, the filter-level pruned models can adequately enjoy the benefits of decreased convolution filters, providing realistic performance improvements. For intelligent edge devices, their run-time systems are usually closed-source or unmodifiable, limiting the applicability of framework-dependent optimizations such as low-precision quantization [14]. In addition, some of these optimizations also introduce extra computation overheads. In contrast, the models optimized by filter pruning will be independent from specific frameworks or libraries, exhibiting better applicability and generality.

Nevertheless, the existing filter pruning methods [11]-

[13] focus only on optimizing models by pruning parametric operators such as convolution, as shown in Figure 1. The non-parametric operators like an element-wise operation are generally considered to be unimportant on the general-purpose platforms such that they are rarely considered by neural network pruning optimization, whereas it is a different story on specialized intelligence platforms. In this paper, we analyze the fine-gained operator-level performance of DNN models, rather than end-to-end performance in previous studies [4], and find that a non-parametric operator also plays an important role when executed on the intelligent edge devices due to its timeconsuming process (Section III), which has, however, been neglected before. To accelerate DNN models on intelligent edge devices effectively, it is thus imperative to take the characteristics of specialized intelligence hardware into account.

In this paper, we present FUPRUNER (Fusion-catalyzed Pruner), a novel optimization approach that, unlike prior work, prunes non-parametric and parametric operators simultaneously, aiming to accelerate the neural network inference on intelligent edge devices with as little accuracy loss as possible. The key idea is a new aggressive fusion scheme for enlarging the optimization space for pruning, by transforming existing operators and inserting auxiliary ones while preserving the equivalence of neural network models, so that non-parametric operators can be pruned in the same manner as parametric operators. Moreover, FUPRUNER supports configurable optimizations, including fusion option and pruning rate to enable flexible trade-offs between the accuracy loss and inference performance, making it possible to optimize models while meeting the accuracy requirement. To the best of our knowledge, FUPRUNER is the first neural network pruning approach with the ability to prune non-parametric operators.

In summary, this paper makes the following contributions:

- We propose a novel fusion-catalyzed pruning approach, namely FUPRUNER, to accelerate deep neural networks on intelligent edge devices by pruning parametric and non-parametric operators simultaneously.
- Our pruning technique facilitates flexible optimization configurations, which allows users to optimize a model in accordance with its requirements, thereby realizing reasonable accuracy-performance trade-offs.
- We demonstrate the effectiveness and efficiency of FUPRUNER by optimizing state-of-the-art deep neural networks on CIFAR-10 and ImageNet datasets.

II. BACKGROUND AND RELATED WORK

In recent years, edge intelligence [2] has become more and more popular, which benefits from the unprecedented success of deep learning, making the research on efficient ondevice inference becomes an inevitable trend. In this section, we first introduce the existing optimization technologies for accelerating DNN models from the perspectives of software down to hardware and then review the work most closely related to this paper, namely, neural network pruning (Section II-A) and operator fusion (Section II-B).

 Model Compression. Recent studies on model compression techniques, including neural network pruning [5] and lowprecision quantization [14], modify neural networks by reducing redundancy, thereby realizing inference acceleration. In general, re-training processes are required for maintaining the accuracy of optimized models. The optimized model parameters are reconstructed from the original ones, and the model compression can thus be regarded as an optimization technique in the model design and development stage.

- Inference Frameworks. Existing deep learning frameworks for on-device inference, including TensorFlow Lite [15], PyTorch Mobile [16], NCNN [17], and MNN [18], are equipped with the ability to analyze and optimize the trained models by using system-level optimization (e.g., data layout selection, operator fusion and parallelization) in the deployment stage. The major difference between the optimizations in these inference frameworks and the model compression is that the former usually focus only on implementation without modifying the model itself.
- Specific Hardware. To solve the problem of inefficiency on general-purpose hardware platform (e.g., CPU), many intelligent edge devices with specialized hardware exist, including Google Edge TPU [6], Intel NCS [7], NVIDIA Jetson TX2 [19], and NVIDIA Jetson Nano [20]. In addition, several studies, such as VTA [21], deployed customhardware designs based on FPGAs. While DNN models can be deployed on these inteligent edge platforms by specific runtime systems, which are usually closed-source or unmodifiable, the model compression techniques can be used to further optimize the inference performance.

Besides, there are several studies on the adaptive inference for optimizing deep learning on embedded platforms, including adaptive strategies for neural network inference [22]–[25] and hardware/software co-design [26]–[28], which allow deep neural networks to be configurable and executed dynamically at runtime based on the resource constraints.

Our approach represents the first for model compression, which essentially optimizes the DNN model by pruning and retraining. Especially, we have designed an aggressive operator fusion scheme to solve new problems in pruning domains and our proposed approach is applied for accelerating models on some specific hardware finally. The proposed approach has two major advantages over the prior work. First, FUPRUNER represents a framework-independent approach, increasing its applicability in practice, as it requires neither special implementations nor run-time system modifications. Second, FUPRUNER does not incur any extra run-time overhead, make it well suited to resource-constrained edge platforms.

A. Neural Network Pruning Methods

Unlike the early efforts on weight pruning methods [8]–[10] that may cause the unstructured sparsity, filter pruning methods shrink a DNN model into a thinner one by reducing the redundant convolutional filters or channels, making structured sparsity for optimized models to achieve realistic acceleration. Li *et al.* [29] pruned unimportant filters with small ℓ_1 -norm. He *et al.* [12] performed a channel-level pruning based on LASSO regression and least square reconstruction. Lou *et al.* [11] selected the filter to be pruned according to the statistics

3

information. In addition to above studies on pruning filters directly in an unrecoverable manner, recently, He *et al.* [13], [30] proposed a "soft pruning manner" by dynamically pruning filters during training, which enlarges the optimization space and remains the model capacity, enabling the pruned filters to be potentially recovered, thereby realizing higher accuracy. Although the pruning methods are advocated in accelerating models, the existing approaches focus only on parametric operators (e.g., convolution). In this paper, our proposed approach can prune non-parametric and parametric operators simultaneously, leading to more efficiency for neural network inference on intelligent edge devices. To the best of our knowledge, FUPRUNER is the first pruning approach with the ability of optimizing non-parametric operators.

B. Operator Fusion Techniques

In general, deep learning frameworks represent a neural network architecture as a graph-based intermediate representation (IR) [31]–[33], known as a computation graph. Operator fusion is a commonly used optimization [34], which fuses two or more adjacent operators into a larger operator with coarser granularity so that the overheads can be reduced as well as the further low-level optimization can be facilitated. For example, it can eliminate the intermediate results incurred by executing a lot of fine-grained operators, reducing the overheads of data transformation and kernel invocation. Current deep learning frameworks, including TensorRT [35], Tensor-Flow [36], MXNet [37], and TVM [38], perform operator fusion for a computation graph by using rule-based strategies repetitively, by reducing gradually the number of operators to be performed. Besides, Jia et al. proposed MetaFlow [39] and TASO [40] to further explore the optimization space on graph substitutions. The fusion rules are formally verified in TASO, thereby achieving not only efficient but also correct model optimizations at the system level. Currently, most of the operator fusion techniques in the inference framework are performed in the deployment stage, i.e., they are used to optimize the models compressed in the development stage. In this paper, we propose a new operator fusion scheme to improve the pruning process from the perspective of model compression, which enables FUPRUNER to optimize nonparametric operators similarly as parametric operators.

III. MOTIVATION

A deep neural network model is composed of various operators and different kinds of operators have different performance characteristics due to different ways in which hardware resources are used. Considering whether to utilize the training process to learn the weights, the DNN operators can be roughly divided into two categories: parametric operators and non-parametric operators. On the one hand, for parametric operators, the convolution operator (abbreviated as COP) is arguably the most important component in contemporary intelligent applications, which is utilized to perform feature extraction. On the other hand, the non-parametric operators that are mainly based on scalar calculation (abbreviated as SOP), such as activation and element-wise addition, also



Fig. 2. The performance of the ResNet-18 model on different devices. The model mainly consists of convolution (conv), batch normalization (bn), element-wise addition (add), and ReLU activation (relu) operators.

provide indispensable functions. Neural network pruning is a key optimization technique for edge intelligence, which accelerates the inference by decreasing the computational cost of operators. However, existing pruning methods mostly focus only on optimizing COPs rather than SOPs, because the COPs are relatively more time-consuming and take up most of the execution time. The performance of SOPs is unattended, even though the non-parametric operators are non-negligible for the DNN inference on intelligent edge devices in practice.

Let $S(\overline{S})$ be the set of pruned (non-pruned) operators in a DNN model. The speedup achieved by a pruned model is:

$$Speedup = \frac{1}{(1 - p(S)) + \frac{p(S)}{a(S)}}$$
(1)

where p(S) is the percentage of the total execution time spent on executing the operators in S, which implies that $p(\bar{S}) = 1 - p(S)$, and a(S) is the speedup achieved for accelerating S. Consequently, the best speedup is limited by Amdahl's law:

$$Speedup \leqslant \frac{1}{1 - p\left(S\right)} = \frac{1}{p(\bar{S})} \tag{2}$$

This shows that the speedup is always limited by $p(\overline{S}) = 1 - p(S)$, which cannot benefit from the pruning optimization.

While preliminary studies have characterized the end-toend performance for neural network models on intelligent edge devices [4], their performance characteristics (e.g., the ratio of its execution time over the total) remain unclear. As an example, we compare the operator-level performance between general-purposed devices (Intel Xeon CPU and ARM Cortex CPU) and specialized devices (Jetson TX2, Jetson Nano) by executing the ResNet-18 [1] on the ImageNet dataset [41]. Figure 2 depicts the performance of ResNet-18 model on different devices. As can be seen, for CPU platforms, concentrating on pruning COPs is a reasonable choice to accelerate DNN models because the $p(\bar{S})$ is very small. However, for intelligent edge platforms, optimizing



Fig. 3. Overview of FUPRUNER. Each arrow represents an optimization step and the dotted block in the same color indicates the corresponding optimized result. (a) to (d) describe the step-by-step optimization sequence that fuses and prunes the "building block #1", (d) to (e) describe the pruning optimization for non-fused "building block #2", and (f) shows the final optimized DNN model.

SOPs can also achieve considerable acceleration due as p(S) becomes non-negligible, which leads to new optimization opportunities. Although the deep learning frameworks usually optimize SOPs at the system level, there are few efforts on optimizing non-parametric operators by the optimization of model compression (e.g., pruning) in the development stage. In this paper, we address the following challenging problem: *Can we optimize non-parametric operators of neural networks in the model compression approach in a similar manner as for parametric operators?*

IV. PROPOSED APPROACH

In this section, we will give a comprehensive introduction of FUPRUNER, and highlight the key novelty of our approach: pruning neural networks catalyzed by operator fusion.

A. Overview

Figure 3 gives the workflow of our approach and shows that how we can optimize non-parametric operators using the pruning process. Given a pre-trained deep neural network model, FUPRUNER will optimize it in terms of user-defined parameters, fusion options and pruning rates. The fusion option determines the building blocks to be fused (marked with a " \square "), and the pruning rate (p_{φ}) determines the number of filters to be deleted for operators. These parameters control the degree of optimization being applied, enabling the accuracy and performance tradeoffs to be made. With reference to $\mathbb{1}$ - $\mathbb{1}$ in the figure, we summarize our optimization steps as follows:

- Inserting and Transforming Operators. The nonparametric operators in the original model are difficult to be fused, which is limited by their different computation modes from the parametric operators. To extend the opportunities of optimization for operator fusion, we insert or transform some operators in the original model, while maintaining the equivalence before and after the model transformation.
- ② Fusing Operators. Unlike existing operator fusion methods that focus on decreasing the model's computational cost, we utilize operator fusion techniques to merge non-parametric operators into parametric operators so that the

non-parametric operators can be pruned in later steps, which, however, may degrade the performance due to the operators inserted or transformed in step ①. Note that the architecture of the fused model is different from the original one, whereas they are equivalent.

- ③ Pruning Fused Operators. The auxiliary operators and non-parametric operators are fused into other operators in the fused model, leading to the dilated weights of the corresponding fused operators, i.e., the fused operators have more filters than before. Therefore, we perform a filter pruning process to decrease the number of such filters.
- **④ Pruning Whole Model.** The other operators except for the fused operators will be pruned according to the rate p_{φ} , so that a much smaller optimized model is obtained. We prune not only parametric operators but also non-parametric operators in the final optimized model by combining filter pruning with operator fusion techniques.

B. Aggressive Operator Fusion

Existing operator fusion techniques usually perform a diminishing fusion approach, which gradually reduces the number of neural network operators by fusing and transforming the execution order of operators if necessary. On the one hand, these fusion methods are performed only in the deployment stage, whereas integrating the fusion approach into the model compression in the development stage may facilitate the model optimization. On the other hand, the conservative fusion scheme may miss the opportunities for further optimizing models. In this paper, we have designed an aggressive fusion scheme, which relies on inserting auxiliary operators and transforming existing operators while keeping the equivalency, to explore more optimization spaces, enabling the possibility of pruning non-parametric operators.

1) Inserting and Transforming Convolution Operators: We define two auxiliary convolution operators: an auxiliary convolution operator for channel-wise fusion (denoted as xconv-C) and an auxiliary convolution operator for filter-wise fusion (denoted as xconv-F). xconv-C, which is designed for the concatenation structures, merges the output of a convolution



Fig. 4. Inserting auxiliary convolution operators: (a) the inserting and fusing procedure of xconv-C, and (b) the procedure of xconv-F.

with another data, and $xconv-\mathcal{F}$, which is responsible for the element-wise addition structures, performs addition of the output of a convolution and another data, as shown in Figure 4. new conv denotes a fused convolution operator, which is generated by the operators in dotted boxes. We use the concatenation structure as an example to demonstrate the approach for channel-wise fusion. The input of the structure is denoted by x and the output by y. To achieve the equivalence, the weights of the auxiliary convolution operators are composed by specific identity matrices (denoted as W), so that x =xconv(x, W). First of all, we can represent the original structure as y = concat(conv(x, W), x). Then, we insert a *xconv*-C into the structure: y = concat(conv(x, W), xconv(x, W)). By their definitions, we can fuse conv and xconv-C into a $new \ conv$ along the dimension of C, and consequently, obtain the fused structure: y = new conv(x, concat(W, W)). As such, despite the introduced operators, the output of the neural network remains unchanged. The approach for fusing addition structures, by fusing add into new conv in element-wise addition structures, is similar. The major difference between filter-wise and channel-wise fusion lies in the dimension of the concatenation operation for fusing the convolution operators. In addition, the input of new conv in the fused elementwise addition structure is a concatenation of the two original *input*'s. In practice, we can optimize common structures in deep neural networks by leveraging x conv-C and x conv-Fcooperatively, as discussed below.

Figure 5 depicts the approach for fusing a basic residual block, which is a popular structure in state-of-the-art neural networks, by inserting auxiliary convolution operators. We use $k \times c \times r \times s$ to represent the configuration of a convolution operator, where k denotes the number of filters, c denotes the channels, r denotes the height of filters, and s denotes the width of filters. There are two 3×3 convolution operators and a shortcut from input to the element-wise addition operator in the original structure. In order to be fusible, the filter size of auxiliary operators must be the same as that for the existing convolution operators, i.e. 3×3 . Moreover, C in xconv-C is the same as that for the first convolution operator and K in xconv-F is the same as the second convolution operator. Then, the convolution and auxiliary operators can be fused along the dimension of C or K, as shown in Figure 5(c). Thus, the



Fig. 5. An example of aggressive fusion for a basic residual block: (a) original structure; (b) inserting auxiliary convolution operators (highlighted in yellow); (c) fusing original operators and auxiliary operators to new operators.



Fig. 6. An example of aggressive fusion for a residual block with a projection shortcut: (a) original structure; (b) inserting auxiliary convolution (highlighted in yellow) and padding convolution operators (highlighted in orange); (c) fusing original operators and auxiliary operators to new operators.

element-wise addition operator in the structure is reduced.

By inserting auxiliary convolution operators, the shortcut connection can be reduced. However, sometimes there are operators on the connection, hindering their applicability. As such, we propose padded convolution operators, *pconv*-C and *pconv*-F for channel-wise and filter-wise fusion. The padded convolution operator is transformed from an existing convolution operator in neural networks by padding weights. For example, a 1×1 convolution operator can be transformed to a 3×3 convolution operator by padding weights with zeros and remains to produce the same output.

Figure 6 depicts the approach for fusing a residual block with projection shortcut by inserting auxiliary convolution operators and transforming padded convolution operators. There is a 1×1 convolution operator on the shortcut, which cannot directly be fused with other operators. As such, we transform the 1×1 convolution operator to a 3×3 padded convolution operator, *pconv*- \mathcal{F} , which can be fused with the second original 3×3 convolution operator (green area). After that, the fusion of the structure can be completed in a similar way to the basic residual block.

2) Adjusting Batch Normalization Operators: Batch normalization [42], a useful technique that standardizes the inputs for each mini-batch, has been widely used in contemporary deep neural networks, which can improve the performance and stability. Generally, the batch normalization operator (abbreviated as bn) can be formalized as:

$$BN(x) = \omega x + \lambda \tag{3}$$

where

$$\omega = \frac{\gamma}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{4}$$



Fig. 7. Disposing batch normalization operators. (a) shows the adjusting procedure of batch normalization operators with xconv-C, and (b) shows the procedure with xconv-F.

$$\lambda = \beta - \omega \cdot \mu_{\mathcal{B}} \tag{5}$$

The $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are the mean and variance for a mini-batch, and the ϵ is a constant for numerical stability. The γ and β are the learned parameters for scale and shift. The batch normalization can be seen as an affine transformation and the detailed algorithm can be seen in [42]. The architecture of neural network that contains bn operators is more complex to achieve an equivalence transformation.

Figure 7 depicts the procedure of disposing batch normalization operators. For the channel-wise fusion, an auxiliary bn operator ($\tilde{\omega} = 1$ and $\tilde{\lambda} = 0$) is inserted, which will not influence the result. Then, the original bn operator and auxiliary bn operator can be fused into a *new* bn operator. For the filter-wise fusion, the bn operator remains unchanged in the fused block to achieve equivalency. However, the bnoperator will disturb the result of xconv- \mathcal{F} . As such, we update the weights of xconv- \mathcal{F} by using the parameters of bn(i.e., perform a inverse operation of bn on the \tilde{W}). In this way, the aggressive fusion method can be applied to the structures which have bn operators.

C. Dynamic Filter Pruning

The fusion process is an equivalence transformation (i,e,. the models before and after fusion are equivalent), enabling the opportunity of pruning SOPs in the original model by fusing some SOPs into COPs. The fused model reduces the SOPs of the model but the COPs are dilated, which is because the weights of auxiliary operators are merged into original operators. Fortunately, we note that the involved weights in auxiliary convolutions and padded convolutions are dramatically sparse, which have more potential for compression. As such, the fused model will be pruned to reduce the redundant weights. In FUPRUNER, we use a soft filter pruning scheme [13], which prunes the filters dynamically. Unlike the classical static pruning scheme, the pruned filters in dynamic pruning scheme can be updated and recovered in the training stage, which has more model capacity and higher accuracy.

We divide the pruning process into two situations: 1) we prune the operators that involve new weights by the aggressive

Algorithm 1: Dynamic Filter Pruning
Input: X (training data), p_{φ} (pruning rate),
e_{max} (training epoch), W (original weights)
Output: W^* (pruned weights of model)
1 for $i \leftarrow 1$ to e_{max} do
2 Update the parameters W by using data X ;
3 foreach $op \in model$ do
4 if $op.type = COP$ then
5 $n \leftarrow$ the filter number of op ;
6 Calculate the ℓ_2 -norm for the filters;
7 if op is fused then
8 $m \leftarrow$ the orginal filter number of <i>op</i> ;
9 Zeroize the lowest $n - m$ filters;
10 else
11 Zeroize the lowest $p_{\varphi} \times n$ filters;
\square

12 Obtain the pruned model parameters W^* from W; 13 return W^* ;

fusion, which prunes the model to recover the original size, i.e., the weights involved by fusion are pruned after basic pruning and the SOPs are reduced; 2) we use a pruning rate p_{φ} to prune the weights of non-fused COPs. As such, the COPs and SOPs of the neural network model are optimized simultaneously. For each training epoch, we dynamically select the filters to be pruned according to their ℓ_2 -norm:

$$\|W_k\|_2 = \sqrt{\sum_{t=1}^{c} \sum_{i=1}^{r} \sum_{j=1}^{s} w_{t,i,j}^2}$$
(6)

where W_k denotes the k-th filter in a convolution operator, and $w_{t,i,j}$ denotes an element of W_k that resides in the *i*-th row and *j*-th column in the *t*-th input channel.

The dynamic filter pruning approach is summarized in Algorithm 1. For each epoch, the weight parameters W are updated by using training data (Line 2). Then, for each COP, we obtain its filter amount n and calculate the ℓ_2 -norm for all filters (Lines 5-6). The filters with low ℓ_2 -norm will be pruned (masked by the zero matrices). The fused operators are pruned out of n-m filters (Lines 8-9), where m indicates the filter amount of the original operators not fused, while the non-fused operators are pruned out of $p_{\varphi} \times n$ filters (Line 11). For dynamic filter pruning, the weights of pruned filters can still be updated in the next epoch, which retains the learning capability. Finally, the pruned model parameters W^* are obtained from the updated W according to the mask information in the last training epoch, i.e, the masked filters are removed (Lines 12-13).

V. EXPERIMENTAL SETUP

Evaluation Platforms. We perform the procedure of fusion and pruning on a cloud server with Intel Xeon CPUs and an NVIDIA Tesla V100 GPU. We evaluate the performance of our approach on five representative intelligent edge devices, which contain different machine learning (ML) acceleration modules, including Jetson TX2, Jetson Nano, Coral Dev Board (Edge TPU), Movidius Neural Compute Stick (NCS),

Category	GPU-Based	Edge Devices	ASIC-Based Edge Devices					
Platform	Jetson TX2 [19]	Jetson Nano [20]	Coral Dev Board (Edge TPU) [6]	Movidius Neural Compute Stick [7]	Neural Compute Stick 2 [43]			
CPU	2 Denver & Cortex-A57	Cortex-A57	Cortex-A53	N/A	N/A			
ML Acceleration Module	256-Core Pascal GPU	128-Core Maxwell GPU	Edge TPU	Myriad X VPU	Myriad 2 VPU			
Memory	8 GB	4 GB	1 GB	N/A	N/A			
Inference Framework	Caffe	Caffe	TensorFlow Lite	OpenVINO	OpenVINO			

 TABLE I

 The specifications for the intelligent edge devices used.

and Neural Compute Stick 2 (NCS2). Table I describes the specifications of these intelligent edge devices. The optimized model will be executed on the inference framework supported by the target platform. Especially, the Jetson TX2 and Jetson Nano run Linux-based operating systems. We leverage the Caffe framework [44], a plain deep learning framework, to evaluate the inference performance of models on both, so that can avoid interference of the optimizations other than ours.

Benchmark Datasets. We evaluate our approach on two representative image classification benchmarks: CIFAR-10 [45] and ImageNet (ILSVRC2012) [41]. CIFAR-10 contains 60000 images (50000 for training and 10000 for testing), classified in ten categories. ImageNet is a large-scale dataset, which has more than one million images classified in 1000 categories.

Neural Network Models. In this paper, we focus on optimizing the state-of-the-art residual neural network (ResNet) models [1]. As revealed in previous studies [12], the ResNet models are less redundant and more challenging to compress. Moreover, the shallower versions of ResNet models are selected, which have smaller sizes and less computation, due to the limited resources of edge devices. Table II describes the architectures of the models, including ResNet-20 and ResNet-32 for CIFAR-10, and ResNet-18 and ResNet-34 for ImageNet. In this table, the building blocks are shown in brackets and the succeeding "×number" denotes the numbers of blocks stacked. A convolution operator (conv) is denoted as a tuple of the size and number of its filters, while a fullyconnected operator (fc) is denoted as a number of its output dimensions. Especially, each building block in residual neural networks except conv1 and fc ends with an element-wise addition operator (add), as shown in Figure 5 and Figure 6.

Optimization Setting. The fusion and pruning of FUPRUNER approach are based on the PyTorch framework [46]. In the pruning stage, its default data argumentation strategy is utilized. Meanwhile, we use the same setting of hype-parameters and training schedules as [13]. The fusion options are denoted as "x/n", where n is the number of stages contained in the model, and x is the number of stages to be fused. For simplicity, we set the stages to be fused in order according to the stage numbers. We prune the fused convolution operators of neural networks, which retains exactly the same number of filters as before fusion. We set the same continued pruning rates, from 0 to 0.3, for all non-fused convolution operators. As such, the fusion option and pruning rate are utilized to

 TABLE II

 THE ARCHITECTURES OF THE RESIDUAL NEURAL NETWORKS.

Layer	Structure	20	32	Structure	18	34
conv1	[3×3, 16]	$\times 1$	$\times 1$	[7×7,64]	$\times 1$	$\times 1$
stage1	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \\ add \end{bmatrix}$	×3	×5	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \\ add \end{bmatrix}$	×2	×3
stage2	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \\ add \end{bmatrix}$	×3	×5	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \\ add \end{bmatrix}$	×2	×4
stage3	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \\ add \end{bmatrix}$	×3	×5	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \\ add \end{bmatrix}$	×2	×6
stage4	-	-	-	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \\ add \end{bmatrix}$	×2	×3
fc	[10]	$\times 1$	$\times 1$	[1000]	×1	$\times 1$

balance accuracy and performance.

VI. EVALUATION

In this section, we demonstrate the effectiveness of our proposed approach, which successfully optimizes DNN models with a negligible accuracy loss. Specifically, we focus on answering two research questions:

- **RQ1.** Can FUPRUNER prune the DNN models effectively with a negligible accuracy loss?
- **RQ2.** Can FUPRUNER accelerate the end-to-end inference of deep neural networks on intelligent edge devices?

A. RQ1. The Model Accuracy

We compare FUPRUNER with state-of-the-art acceleration methods, including MIL [47], PFEC [51], SFP [13] and ASFP [30]. For fairness, the accuracy numbers are directly cited from the corresponding papers. While the architecture of a baseline model is the same for all methods, the accuracy numbers are slightly different due to different experimental hyper-parameters, such as learning rate and data augmentation. As such, we report not only the accuracy of optimized models but also the baseline models. We use "Acc. Drop" to illustrate the accuracy dropping of the optimized model compared to the original model. Furthermore, we run each experiment three times and report the result as (mean \pm std), where mean represents the average and std the standard deviation, and all the accuracies of compressed models are evaluated on the cloud server with the same environment.

TABLE III THE ACCURACY OF PRUNED RESIDUAL NEURAL NETWORK MODELS ON THE CIFAR-10 DATASET.

Model	Method	Top-1 Acc. Baseline (%)	Top-1 Acc. Pruned (%)	Top-1 Acc. Drop (%)	Fusion Option	Pruning Rate (%)
	MIL [47]	91.53	91.43	0.10	N/A	N/A
	SFP-0.1 [13]	92.20	92.24	-0.04	N/A	10.0%
	SFP-0.2 [13]	92.20	91.20	1.00	N/A	20.0%
	SFP-0.3 [13]	92.20	90.83	1.37	N/A	30.0%
	FUPRUNER-1/3	92.60 (±0.27)	92.86 (±0.02)	-0.26	1/3	0.0%
ResNet-20	FUPRUNER-2/3	92.60 (±0.27)	92.58 (±0.10)	0.02	2/3	0.0%
	FUPRUNER-3/3	92.60 (±0.27)	92.34 (±0.05)	0.26	3/3	0.0%
	FUPRUNER-1/3-0.1	92.60 (±0.27)	92.77 (±0.19)	-0.17	1/3	10.0%
	FUPRUNER-1/3-0.2	92.60 (±0.27)	92.28 (±0.13)	0.32	1/3	20.0%
	FUPRUNER-1/3-0.3	92.60 (±0.27)	91.65 (±0.16)	0.95	1/3	30.0%
	FUPRUNER-3/3-0.3	92.60 (±0.27)	91.40 (±0.05)	1.20	3/3	30.0%
	MIL [47]	92.33	90.74	1.59	N/A	N/A
	SFP-0.1 [13]	92.63	93.22	-0.59	N/A	10.0%
	SFP-0.2 [13]	92.63	90.63	2.00	N/A	20.0%
	SFP-0.3 [13]	92.63	90.08	2.55	N/A	30.0%
	FUPRUNER-1/3	93.41 (±0.16)	93.29 (±0.09)	0.12	1/3	0.0%
ResNet-32	FUPRUNER-2/3	93.41 (±0.16)	92.40 (±0.09)	1.01	2/3	0.0%
	FUPRUNER-3/3	93.41 (±0.16)	92.32 (±0.08)	1.09	3/3	0.0%
	FUPRUNER-1/3-0.1	93.41 (±0.16)	93.33 (±0.02)	0.08	1/3	10.0%
	FUPRUNER-1/3-0.2	93.41 (±0.16)	93.23 (±0.14)	0.18	1/3	20.0%
	FUPRUNER-1/3-0.3	93.41 (±0.16)	92.55 (±0.26)	0.86	1/3	30.0%
	FUPRUNER-3/3-0.3	93.41 (±0.16)	91.94 (±0.07)	1.47	3/3	30.0%

TABLE IV THE ACCURACY OF PRUNED RESIDUAL NEURAL NETWORK MODELS ON THE IMAGENET DATASET.

Madal	Mathad	Top-1 Acc.	Top-1 Acc.	Top-1 Acc.	Top-5 Acc.	Top-5 Acc.	Top-5 Acc.
Niodei	Method	Baseline (%)	Pruned (%)	Drop (%)	Baseline (%)	Pruned (%)	Drop (%)
	MIL [47]	69.98	66.33	3.65	89.24	86.94	2.3
	SFP-0.3 [13]	70.23	60.79	9.44	89.51	83.11	6.4
	ASFP-0.3 [30]	70.23	68.02	2.21	89.51	88.19	1.32
	XNOR-Net++ [48]	69.30	57.10	12.2	89.20	79.90	9.30
ResNet-18	Bi-Real Net [49]	69.30	56.40	12.9	89.20	79.50	9.70
	CI-BCNN [50]	69.30	59.90	9.40	89.20	84.18	5.02
	FUPRUNER-1/4	69.76	70.97 (±0.03)	-1.21	89.08	89.85 (±0.03)	-0.77
	FUPRUNER-2/4	69.76	70.75 (±0.07)	-0.99	89.08	89.81 (±0.03)	-0.73
	FUPRUNER-3/4	69.76	70.61 (±0.01)	-0.85	89.08	89.70 (±0.06)	-0.62
	FuPruner-4/4	69.76	70.39 (±0.44)	-0.63	89.08	89.55 (±0.17)	-0.47
	FUPRUNER-4/4-0.2	69.76	69.69 (±0.25)	0.07	89.08	89.35 (±0.08)	-0.27
	FUPRUNER-4/4-0.3	69.76	68.24 (±0.45)	1.52	89.08	88.21 (±0.27)	0.87
	MIL [47]	73.42	72.99	0.43	91.36	91.19	0.17
	PFEC [51]	73.23	72.17	1.06	-	-	-
	SFP-0.3 [13]	73.92	72.29	1.63	91.62	90.90	0.72
	ASFP-0.3 [30]	73.92	72.53	1.39	91.62	91.04	0.58
DecNet 24	Bi-Real Net [49]	73.30	62.20	11.1	91.30	83.90	7.40
Keshel-34	CI-BCNN [50]	73.30	64.93	8.37	91.30	86.61	4.69
	FuPruner-1/4	73.32	74.28 (±0.11)	-0.96	91.42	91.90 (±0.07)	-0.48
	FuPruner-2/4	73.32	74.23 (±0.04)	-0.91	91.42	91.76 (±0.04)	-0.34
	FUPRUNER-3/4	73.32	73.85 (±0.03)	-0.53	91.42	91.61 (±0.04)	-0.19
	FuPruner-4/4	73.32	73.10 (±0.05)	0.22	91.42	91.23 (±0.03)	0.19
	FUPRUNER-4/4-0.2	73.32	72.86 (±0.07)	0.46	91.42	91.13 (±0.08)	0.29
	FUPRUNER-4/4-0.3	73.32	72.14 (±0.05)	1.18	91.42	90.66 (±0.02)	0.76

We evaluate our approach on the CIFAR-10 dataset by using ResNet-20 and ResNet-32 models, as shown in Table III. The top-1 accuracy of each method is reported in the table. Besides classical filter pruning methods, we also compare the proposed approach with MIL [47], an acceleration method that involves new efficient structures into the original neural network models. We perform two pruning approaches: 1) conservative pruning: we only prune the filters of dilated convolution operators incurred by aggressive fusion, which is denoted as "FUPRUNER-(fusion option)", according to the fusion option; 2) continued pruning: we further prune non-fused convolution operators based on the conservative pruned models to achieve simultaneous optimization for SOPs and COPs, which is denoted as "FUPRUNER-(fusion option)-(continued pruning rate)". For instance, the "FUPRUNER-1/3" means that we fuse building blocks in the first stage of the neural network and



Fig. 8. Comparing the inference performance results among the original model, the models optimized by FUPRUNER, and the models optimized by classic filter pruning on CIFAR-10. The left-most blue bar denotes the inference time of the original DNN model, the three bars in P1 denote the conservative pruning by FUPRUNER, and the three bars in P2 denote the continued pruning. The above-bar numbers denote the relative speedups over the original model.

then prune the fused convolution operators to recover the size of the model. The "FUPRUNER-1/3-0.3" means that we prune non-fused convolution operators of the neural network with the $p_{\varphi} = 0.3$ based on the model of "FUPRUNER-1/3". Firstly, FUPRUNER can prune SOPs with a negligible accuracy loss. For example, the optimized ResNet-20 model by "FUPRUNER-1/3" fuses and prunes about 1/3 building blocks without any accuracy loss, and the model by "FUPRUNER-3/3", which fuses and prunes all the element-wise addition operators from stage1 to stage3, drops only by 0.26% in accuracy. When performing the continued pruning approach that further prunes COPs, the performance of FUPRUNER also outperforms other state-of-the-art methods. As can be seen, our approach exhibits less accuracy loss compared to other methods for the same pruning rate. The accuracies of some pruned models, such as the ResNet-20 models with "SFP-0.1" [13] and "FUPRUNER-1/3", are even better than the baseline model. This shows that the filter pruning approach has a regularization effect and can reduce the over-fitting of neural network models.

We also evaluate our approach on ImageNet, a large-scale dataset, by using ResNet-18 and ResNet-34 models, as shown in Table IV. We report the top-1 accuracy and top-5 accuracy for each method and the results on ImageNet are similar to those on CIFAR-10. Our approach can prune SOPs without any accuracy loss in most cases. The optimized ResNet-34 model by "FUPRUNER-4/4", which fuses all building blocks, only drops by 0.22% in top-1 accuracy and 0.19% in top-5 accuracy. Furthermore, the models optimized by continued pruning achieve better performance compared to other methods. These experimental results show the effectiveness of FUPRUNER, which prunes SOPs and COPs simultaneously and achieves comparable performance as original models. We have also compared our approach with several state-of-theart neural network binarization methods, including XNOR-Net++ [48], Bi-Real Net [49], and CI-BCNN [50]. While the binarized models enjoy the potential of storage compression and acceleration, their performance degradation is unsatisfactory for real intelligent applications.

B. RQ2. The Realistic Acceleration

We address RQ2 by evaluating the performance of singlebatch inference on representative intelligent edge devices. We compare the original model with the models optimized by FUPRUNER with different optimization settings. The inference time reported of a model is the average of 1000 runs.

1) Results on CIFAR-10. Figure 8 depicts the results on CIFAR-10 dataset. As for optimized models by FUPRUNER, we set fusion option $\in \{1/3, 2/3, 3/3\}$ for conservative pruning (P1), and we set fusion option = 3/3 and $p_{\omega} \in$ $\{0.1, 0.2, 0.3\}$ for continued pruning (P2). Moreover, the classic filter pruning approach with the rate = 0.3 is used as the reference. There are three observations. First, the fused models with conservative pruning outperforms the original models, providing the evidence for superior performance benefits obtainable from non-parametric operator pruning. Second, the optimized models by continued pruning cannot always yield acceleration, and may even lead to performance degradation, with, e.g., the continued pruning results on NCS. This is because the convolution operators with special sizes have more efficient implementations in deep learning frameworks. For example, the convolution operators with 64 filters often perform faster than those with 63 filters, even though the latter have fewer filters, because the regular data size is more conducive to the optimizations such as parallelization. Finally, FUPRUNER outperforms classic filter pruning approaches, which prune only COPs, with the pruning rate = 0.3, in most cases. The classic filter pruning approaches prune a little more convolution operators than our approach (are thus faster on some devices such as Edge TPU), because the fused operators will not continue to be pruned in FUPRUNER.

2) Results on ImageNet. Table V gives the best speedup over the original model for each optimization combination of (fusion option, p_{φ}) in $\{1/4, 2/4, 3/4, 4/4\} \times \{0, 0.1, 0.2, 0.3\}$ on ImageNet. We report some results obtained by FUPRUNER with different optimization parameters, the best optimization parameter values (Best Param.) and their speedups achieved. Moreover, we use the classic filter pruning approach with the pruning rate = 0.3 as a reference in the last column (Ref-0.3). The results demonstrate the flexibility of FUPRUNER, that is,

 TABLE V

 The best speedups of the optimized neural network models on the ImageNet dataset.

		FUPRUNER									
Model	Device		Speedu	ps (for S	Post Daram	Speedup	Speedup				
		1/4	2/4	3/4	4/4	4/4-0.1	4/4-0.2	4/4-0.3	Dest l'alam.	Speedup	Speedup
	Jetson TX2	1.02×	$1.06 \times$	1.09×	1.12×	1.15×	$1.22 \times$	1.33×	3/4-0.3	1.34×	1.30×
ResNet-18	Jetson Nano	1.02×	$1.06 \times$	$1.07 \times$	1.13×	1.14×	1.21×	1.35×	3/4-0.3	1.35×	1.29×
	Edge TPU	1.02×	$1.02 \times$	1.03×	1.03×	1.04×	1.15×	1.30×	1/4-0.3	1.43×	$1.42 \times$
	NCS	1.00×	$1.01 \times$	$1.02 \times$	1.02×	0.18×	$0.22 \times$	0.23×	4/4	1.02 imes	0.21×
	NCS2	1.01×	$1.03 \times$	1.04×	1.06×	1.01×	$1.10 \times$	1.19×	2/4-0.3	1.24×	$1.22 \times$
	Jetson TX2	1.02×	$1.04 \times$	$1.07 \times$	1.09×	1.12×	1.19×	1.29×	2/4-0.3	1.33×	1.33×
ResNet-34	Jetson Nano	1.02×	$1.05 \times$	1.09×	1.10×	1.11×	$1.22 \times$	1.33×	3/4-0.3	1.37×	1.25×
	Edge TPU	1.02×	$1.01 \times$	$1.01 \times$	1.03×	1.02×	1.13×	$1.28 \times$	1/4-0.3	1.46×	1.46×
	NCS	1.00×	$1.01 \times$	1.01×	1.01×	0.19×	0.23×	0.23×	4/4	1.01×	0.21×
	NCS2	1.01×	$1.03 \times$	1.06×	1.07×	$1.04 \times$	$1.15 \times$	$1.25 \times$	2/4-0.3	1.33×	$1.32 \times$

 TABLE VI

 The performance results for the optimized neural network models on the CIFAR-10 dataset.

Madal	del Optimization Top-1]	Inference Times on Jetson TX2 (ms)					Inference Times on Intel NCS2 (ms)				s)		
WIGUEI	1	Param.	Acc. (%)	Sys.	Conv	Add	BN	Others	Total	Sys.	Conv	Add	BN	Others	Total
	H	Baseline	92.60	1.81	2.22	0.74	4.41	0.36	9.55	2.10	0.86	0.54	1.92	0.40	5.83
		1/3	92.86	1.63	2.17	0.51	4.48	0.44	9.22	2.06	0.86	0.34	1.93	0.48	5.67
		2/3	92.58	1.58	2.10	0.25	4.25	0.50	8.67	2.07	0.84	0.17	1.84	0.55	5.47
t-2(ER	3/3	92.34	1.44	1.97	-	3.98	0.57	7.97	2.05	0.82	-	1.79	0.63	5.28
Ne	I N	3/3-0.1	92.41	1.44	1.99	-	4.00	0.56	7.99	2.08	0.81	-	1.91	0.62	5.42
Res	PR	3/3-0.2	91.92	1.42	1.98	-	4.03	0.58	8.01	2.04	0.81	-	1.89	0.62	5.36
	머	3/3-0.3	91.40	1.46	1.98	-	3.98	0.57	7.99	2.05	0.79	-	1.83	0.62	5.30
		3/3*	92.34	0.60	1.86	-	-	0.57	3.03	1.14	0.88	-	-	0.16	2.18
		3/3-0.3*	91.40	0.63	1.89	-	-	0.59	3.11	1.15	0.85	-	-	0.17	2.17
	H	Baseline	93.41	3.32	3.40	1.24	7.02	0.52	15.49	2.22	1.37	0.89	3.02	0.54	8.05
		1/3	93.29	2.73	3.37	0.83	7.06	0.64	14.64	2.19	1.38	0.55	3.01	0.68	7.81
0		2/3	92.40	2.18	3.34	0.43	6.93	0.76	13.63	2.23	1.35	0.27	2.94	0.80	7.58
t-3;	ER	3/3	92.32	1.92	3.23	-	6.71	0.88	12.74	2.19	1.33	-	2.87	0.91	7.31
Ne	NN N	3/3-0.1	92.60	2.40	3.21	-	6.44	0.86	12.91	2.17	1.33	-	3.09	0.92	7.51
Res	IPR	3/3-0.2	92.41	2.46	3.19	-	6.48	0.85	12.98	2.21	1.31	-	3.03	0.90	7.45
	Ъ	3/3-0.3	91.94	2.27	3.17	-	6.49	0.85	12.79	2.18	1.28	-	2.97	0.89	7.33
		3/3*	92.32	0.93	2.95	-	-	0.85	4.73	1.75	1.42	-	-	0.16	3.33
		3/3-0.3*	91.94	0.97	2.88	-	-	0.85	4.70	1.75	1.38	-	-	0.17	3.29

we can get the best optimized model by adjusting the fusion option and pruning rate. Compared with the original model, the optimized models by FUPRUNER achieve significant performance improvements on most evaluated edge platforms. In addition, the accuracy of optimized model by FUPRUNER is higher than the corresponding reference model (Ref-0.3), as revealed in Table IV. As such, the performance of our approach, which has the less accuracy loss, outperforms the classic filter pruning approaches. One observation is that the speedups of the pruned models by continued pruning are more noticeable than those on CIFAR-10, which is attributed to the larger sizes of input images in the ImageNet dataset. These results show that FUPRUNER provides a flexible optimization framework and can achieve realistic performance improvements.

3) Analysis of Optimized Models. Table VI gives the performance results for the optimized models under a range of optimization parameters on CIFAR-10. To this end, we have selected Jetson TX2 (a GPU-based edge device) and NCS2 (an ASIC-based edge device) as the two representative platforms. The total execution time for a neural network model consists of the times spent on the system invocation, denoted as "Sys.", and the forward computation of different operators, including "Conv" (convolution operators), "Add" (element-wise addition operators), "BN" (batch normalization operaters) and "Others". Our results demonstrate the performance improvements achieved from each component for the end-to-end inference. First, the execution time of non-parametric operators such as "Add" is non-negligible on edge platforms, indicating the necessity for optimizing non-parametric operators. Second, our approach has successfully reduced the execution time of "Add" with conservative pruning. The performance of "Conv" can be further improved by continued pruning. For example, the optimized ResNet-32 model with "FUPRUNER-3/3" decreases the execution time of "Add" from 1.24 ms to 0.00 ms on Jetson TX2, meaning that all the add operators in neural networks have been pruned away. Furthermore, we can merge a BN operator into its previous convolution operator by adjusting and updating the weights, a common systemlevel optimization approach [52], with the results of the BNmerged models denoted as "*". There is neither an elementwise addition operator nor a batch normalization operator in the final pruned model, thereby enabling us to reduce significantly the execution time of neural network inference on edge intelligent devices. Overall, our experimental results have

 TABLE VII

 The performance of the fused models without filter pruning.



Fig. 9. Model accuracy regarding different pruning rates. The orange line denotes the original model while the blue line denotes the optimized model.

confirmed the effectiveness of our approach for optimizing non-parametric operators on intelligent edge devices.

C. Ablation Study

To analyze each component of our proposed approach, ablation studies are conducted.

- 1) Operator Fusion without Pruning. Table VII shows the inference time of fused models without pruning on NCS with the original model used as a baseline. The fused model is more time-consuming than the original one due to the extra auxiliary operators fused into, illustrating the necessity of conservative pruning after fusion.
- 2) Verifying Pruning Rates. Figure 9 shows the model accuracy of different pruning rates for ResNet-20 and ResNet-32. The accuracy of the optimized model decreases with the increase of pruning rate and drops observably when the pruning rate is more than 30%. Therefore, we set the filter pruning rate of FUPRUNER from 0 to 0.3 in this paper.
- 3) Selection of the Fused Building Blocks. We compare the accuracy of fusing different building blocks for ResNet-20, as shown in Table VIII. We use "(stage1, stage2, stage3)" to denote the number of fused building blocks for each stage of the model. The accuracy may further improve if we carefully select the building blocks to be fused in FUPRUNER based on hyper-parameter tuning.

VII. DISCUSSION

Generality. This paper focuses on the convolutional neural network, a widely used architecture that plays an important role in contemporary intelligent tasks. The proposed auxiliary operators are currently based on the computational pattern of convolution operations. To extend our work to other scenarios such as optimizing recurrent neural networks, we need to

 TABLE VIII

 The top-1 accuarcy of ResNet-20 with different fused blocks.

Fused Building	Blocks	Top-1 Accuracy (%)					
in (stage1, stage2	, stage3)	$p_arphi=0.1$	$p_arphi=0.2$	$p_arphi=0.3$			
Fusing 3/9	(3,0,0)	92.93	92.15	91.46			
Building Blocks	(1,1,1)	92.40	92.46	91.83			
Fusing 6/9	(3,3,0)	92.54	92.12	91.47			
Building Blocks	(2,2,2)	93.04	92.30	91.84			

analyze the performance characteristics of the target model on edge devices and design a new set of auxiliary operators for the structures to be optimized. Nevertheless, FUPRUNER provides a new generic framework for optimizing non-parametric operators in the model compression approach similarly as for parametric operators, which can be generalized to support other structures and model acceleration domains.

Applicability. FUPRUNER represents a framework-independent approach, making it more generally applicable than other algorithm-level acceleration approaches, as it requires neither special implementations nor run-time system modifications. Moreover, FUPRUNER does not incur any extra run-time overhead, making it well suited for resource-constrained edge devices. To determine that whether a DNN model on a target device can be accelerated by our approach or not, we can start with an analysis of the performance characteristics of the model on the device. To achieve a realistic acceleration in the optimized model by Equation 1, the time proportion of non-parametric operators in end-to-end inference must be non-negligible. As revealed in our experimental results, the performance characteristics of neural networks on intelligent edge devices are different from those on general-purposed platforms, providing more optimization opportunities for pruning non-parametric operators. We envision for FUPRUNER to become a useful acceleration technique for future edge intelligence.

Soundness. The transformation of a computational graph is equivalence-preserving in the aggressive fusion method, without causing any loss of accuracy, as described in Section IV-B. However, the accuracy of a model may possibly decrease when applying our catalyzed-pruning approach to achieve both acceleration and compression. In this case, FUPRUNER provides configurable optimization options for controlling fusion and pruning, allowing performance/accuracy trade-offs to be made. Limitations. Although FUPRUNER shows the benefits of pruning parametric and non-parametric operators, there is room for further improvements. First, we observe that pruning parametric or non-parametric operators has no significant optimization effect on some platforms, such as NCS, and pruning filters may even lead to performance degradation. Therefore, in addition to focusing on the model compression itself, it is meaningful to design a mechanism to determine whether the compressed model has realistic acceleration. Second, several hyper-parameters, such as fusion option and pruning rate, are necessary and predefined by the user in our approach so that a tuning process is needed currently. We will explore how to automatically set the reasonable optimization hyperparameters. Finally, the accuracy of optimized models mainly depends on the filter pruning algorithm used and our aggressive operator fusion method can be combined with any filter pruning algorithm. Therefore, using different pruning strategies for different fused models is another future research direction.

VIII. CONCLUSION

In this paper, we have introduced FUPRUNER, a novel fusion-catalyzed pruning approach to optimize deep neural network models, which prunes parametric and non-parametric operators simultaneously for a faster inference on intelligent edge devices. Evaluation with state-of-the-art residual neural networks on representative edge platforms has demonstrated the effectiveness of our approach. We would like to apply FUPRUNER to more complex intelligent applications, such as object detection, in the future work.

REFERENCES

- K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision* and pattern recognition, 2016, pp. 770–778.
- [2] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.
- [3] J. Cheng, P.-s. Wang, G. Li, Q.-h. Hu, and H.-q. Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Frontiers* of Information Technology & Electronic Engineering, vol. 19, no. 1, pp. 64–77, 2018.
- [4] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in *IEEE International Symposium on Workload Characterization*, 2019, pp. 1–14.
- [5] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" arXiv preprint arXiv:2003.03033, 2020.
- [6] Google. (2020) Edge TPU. [Online]. Available: https://cloud.google.c om/edge-tpu
- [7] Intel. (2020) Intel movidius neural compute stick. [Online]. Available: https://software.intel.com/en-us/articles/intel-movidius-neural-compute -stick
- [8] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information* processing systems, 2015, pp. 1135–1143.
- [9] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in Advances in neural information processing systems, 2016, pp. 1379–1387.
- [10] X. Dong, L. Liu, G. Li, P. Zhao, and X. Feng, "Fast CNN pruning via redundancy-aware training," in *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 3–13.
- [11] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.
- [12] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.
- [13] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the* 27th International Joint Conference on Artificial Intelligence, 2018, pp. 2234–2240.
- [14] Y. Guo, "A survey on methods and theories of quantized neural networks," arXiv preprint arXiv:1808.04752, 2018.
- [15] Google. (2020) TensorFlow Lite. [Online]. Available: https://www.tens orflow.org/lite/
- [16] Facebook. (2020) PyTorch mobile. [Online]. Available: https: //pytorch.org/mobile/home/
- [17] Tencent. (2020) NCNN. [Online]. Available: https://github.com/Tencent/ncnn
- [18] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu *et al.*, "MNN: A universal and efficient inference engine," *arXiv preprint arXiv:2002.12418*, 2020.
- [19] NVIDIA. (2020) Jetson TX2 developer kit and modules. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embe dded-systems/jetson-tx2

- [21] T. Moreau, T. Chen, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "VTA: an open hardware-software stack for deep learning," *arXiv preprint arXiv:1807.04188*, 2018.
- [22] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2016, pp. 475–480.
- [23] J. Yu, L. Yang, N. Xu, J. Yang, and T. S. Huang, "Slimmable neural networks," in 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 2019.
- [24] P. Panda, A. Ankit, P. Wijesinghe, and K. Roy, "FALCON: Feature driven selective classification for energy-efficient image recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits* and Systems, vol. 36, no. 12, 2017.
- [25] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," ACM Transactions on Embedded Computing Systems (TECS), vol. 19, no. 2, pp. 1–28, 2020.
- [26] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). IEEE, 2016, pp. 1–10.
- [27] N. K. Jayakodi, A. Chatterjee, W. Choi, J. R. Doppa, and P. P. Pande, "Trading-off accuracy and energy of deep inference on embedded systems: A co-design approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2881– 2893, 2018.
- [28] N. K. Jayakodi, S. Belakaria, A. Deshwal, and J. R. Doppa, "Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models," ACM Transactions on Embedded Computing Systems (TECS), vol. 19, no. 4, pp. 1–24, 2020.
- [29] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," arXiv preprint arXiv:1608.08710, 2016.
- [30] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang, "Asymptotic soft filter pruning for deep convolutional neural networks," *IEEE transactions* on cybernetics, pp. 1–11, 2019.
- [31] S. Cyphers, A. K. Bansal, A. Bhiwandiwalla, J. Bobba, M. Brookhart, A. Chakraborty, W. Constable, C. Convey, L. Cook, O. Kanawi *et al.*, "Intel ngraph: An intermediate representation, compiler, and executor for deep learning," *arXiv preprint arXiv:1801.08058*, 2018.
- [32] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein *et al.*, "Glow: Graph lowering compiler techniques for neural networks," *arXiv preprint arXiv:1805.00907*, 2018.
- [33] X. Dong, L. Liu, P. Zhao, G. Li, J. Li, X. Wang, and X. Feng, "Acorns: A framework for accelerating deep neural networks with input sparsity," in 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT). IEEE, 2019, pp. 178–191.
- [34] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, and D. Qian, "The deep learning compiler: A comprehensive survey," arXiv preprint arXiv:2002.03794, 2020.
- [35] H. Vanholder, "Efficient inference with TensorRT," 2016.
- [36] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.
- [37] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," arXiv preprint arXiv:1512.01274, 2015.
- [38] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: end-to-end optimization stack for deep learning," *arXiv preprint arXiv:1802.04799*, 2018.
- [39] Z. Jia, J. Thomas, T. Warszawski, M. Gao, M. Zaharia, and A. Aiken, "Optimizing dnn computation with relaxed graph substitutions," in *Proceedings of the 2nd Conference on Systems and Machine Learning* (SysML'19), 2019.
- [40] Z. Jia, O. Padon, J. Thomas, T. Warszawski, M. Zaharia, and A. Aiken, "TASO: optimizing deep learning computation with automatic generation of graph substitutions," in *Proceedings of the 27th ACM Symposium* on Operating Systems Principles, 2019, pp. 47–62.
- [41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in 2009 IEEE conference on computer vision and pattern recognition. IEEE, 2009, pp. 248–255.

- [42] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," arXiv preprint arXiv:1502.03167, 2015.
- [43] Intel. (2020) Intel neural compute stick 2. [Online]. Available: https://software.intel.com/en-us/neural-compute-stick
- [44] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," arXiv preprint arXiv:1408.5093, 2014.
- [45] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
- [46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.
- [47] X. Dong, J. Huang, Y. Yang, and S. Yan, "More is less: A more complicated network with less inference complexity," in *Proceedings* of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5840–5848.
- [48] A. Bulat and G. Tzimiropoulos, "Xnor-net++: Improved binary neural networks," in 30th British Machine Vision Conference (BMVC), 2019, p. 62.
- [49] Z. Liu, W. Luo, B. Wu, X. Yang, W. Liu, and K.-T. Cheng, "Bireal net: Binarizing deep network towards real-network performance," *International Journal of Computer Vision*, vol. 128, no. 1, pp. 202–219, 2020.
- [50] Z. Wang, J. Lu, C. Tao, J. Zhou, and Q. Tian, "Learning channel-wise interactions for binary convolutional neural networks," in *Proceedings* of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 568–577.
- [51] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in 5th International Conference on Learning Representations, 2017, pp. 1–13.
- [52] D. Kang, E. Kim, I. Bae, B. Egger, and S. Ha, "C-GOOD: C-code generation framework for optimized on-device deep learning," in *Proceedings* of the International Conference on Computer-Aided Design, 2018, pp. 1–8.