RT3D: Achieving Real-Time Execution of 3D Convolutional Neural Networks on Mobile Devices

Wei Niu[1*], **Mengshu Sun**[2*], **Zhengang Li**[2*], **Jou-An Chen**[3], **Jiexiong Guan**[1], **Xipeng Shen**[3], **Yanzhi Wang**[2], **Sijia Liu**[4], **Xue Lin**[2], **Bin Ren**[1]

[1] William & Mary, [2] Northeastern University, [3] North Carolina State University, [4] MIT-IBM Watson AI Lab

[1]{wniu, jguan}@email.wm.edu, [2]{sun.meng, li.zhen, yanz.wang, xue.lin}@northeastern.edu,

[3]{jchen73, xshen5}@ncsu.edu, [4]sijia.liu@ibm.com, [1]bren@cs.wm.edu

Abstract

Mobile devices are becoming an important carrier for deep learning tasks, as they are being equipped with powerful, high-end mobile CPUs and GPUs. However, it is still a challenging task to execute 3D Convolutional Neural Networks (CNNs) targeting for real-time performance, besides high inference accuracy. The reason is more complex model structure and higher model dimensionality overwhelm the available computation/storage resources on mobile devices. A natural way may be turning to deep learning weight pruning techniques. However, the direct generalization of existing 2D CNN weight pruning methods to 3D CNNs is not ideal for fully exploiting mobile parallelism while achieving high inference accuracy.

This paper proposes RT3D, a model compression and mobile acceleration framework for 3D CNNs, seamlessly integrating neural network weight pruning and compiler code generation techniques. We propose and investigate two structured sparsity schemes i.e., the vanilla structured sparsity and kernel group structured (KGS) sparsity that are mobile acceleration friendly. The vanilla sparsity removes whole kernel groups, while KGS sparsity is a more fine-grained structured sparsity that enjoys higher flexibility while exploiting full on-device parallelism. We propose a reweighted regularization pruning algorithm to achieve the proposed sparsity schemes. The inference time speedup due to sparsity is approaching the pruning rate of the whole model FLOPs (floating point operations). RT3D demonstrates up to $29.1\times$ speedup in end-to-end inference time comparing with current mobile frameworks supporting 3D CNNs, with moderate $1\% \sim 1.5\%$ accuracy loss. The end-to-end inference time for 16 video frames could be within 150 ms, when executing representative C3D and R(2+1)D models on a cellphone. For the first time, real-time execution of 3D CNNs is achieved on off-the-shelf mobiles.

1 Introduction

Since the Convolutional Neural Networks (CNNs) were exemplified by the performance improvements obtained by AlexNet (Krizhevsky, Sutskever, and Hinton 2012) in 2012, neural network based computer vision has achieved superhuman performance. Mobile devices are becoming an important carrier for deep learning tasks. However, real-time execution is the most critical requirement given computation/storage resource constraints on mobiles for deep learning tasks.

Recently, many efforts (Han et al. 2016; Yao et al. 2017; Huynh, Lee, and Balan 2017; Chen et al. 2018; Alibaba 2020; TensorFlow 2017; PyTorch 2019) aim to accelerate CNN execution on off-the-shelf mobile devices and some of them achieve significant advancements. However, most of these optimizations focus on traditional 2D CNNs in the image domain. On the other hand, 3D CNNs have been proposed for video domain tasks such as video classification, and action recognition/detection (Ji et al. 2012; Wang et al. 2017; Carreira and Zisserman 2017; Qiu, Yao, and Mei 2017; Köpüklü et al. 2019) It is still an open problem to execute 3D CNNs on off-the-shelf mobile devices targeting for real-time performance. For example, C3D (Tran et al. 2015), a mainstream 3D CNN takes over 2.5 seconds to complete the inference (of 16 frames) on a representative mobile CPU (Kryo 585 in Qualcomm Snapdragon platform) with Pytorch Mobile (PyTorch 2019), which is clearly far from real-time execution.[1] The extra dimension in 3D convolution (CONV) significantly increases storage size and computation workload comparing with 2D CONV.[2] The large memory footprint of 3D CNN models often exceeds the on-chip cache size of off-the-shelf mobile devices. As a result, 3D CNNs are currently supported only by very few mobile acceleration frameworks i.e., PyTorch Mobile (PyTorch 2019) and Alibaba MNN (Alibaba 2020) with relatively low computation efficiency, let alone real-time execution performance.

A natural way to bridge the gap is to turn to *model compression* techniques, particularly *weight pruning* (Wen et al. 2016; Han, Mao, and Dally 2016; Guo, Yao, and Chen 2016; Dong and Yang 2019; Zhuang et al. 2018; Yu et al. 2018; He et al. 2019) which has demonstrated its efficacy on accelerating 2D CNN executions. Nevertheless, generalizing weight pruning methods from 2D to 3D CNNs is more than a straightforward task owing to the higher dimensionality of weight tensors and thus the larger search space of weight pruning. It is especially challenging to derive the best-suited

[*]These authors contributed equally.

[1]Real-time performance requires to compute 30 frames/second according to state-of-the-art industry standard.

[2]2D CONV is a special case of 3D CONV with the temporal dimension size equal to 1.

weight pruning method in order to achieve real-time performance on off-the-shelf mobile devices. Two fundamental problems need to be solved: the *sparsity scheme* and the *pruning algorithm*. The former refers to the *regularity* in pruning, i.e., the specific structural characteristics of CNNs after pruning. The two representative cases for 2D CNNs are the most flexible, irregular pruning scheme that can prune arbitrary weights (Han, Mao, and Dally 2016; Guo, Yao, and Chen 2016), and the computing platform-friendly filter/channel pruning scheme that prunes whole filters/channels (Wen et al. 2016; He et al. 2019; Yu et al. 2018). The latter refers to the appropriate algorithm to determine the target weights to remove and train the remaining, non-zero weights. For 2D CNNs, there is also rich literature in heuristic pruning (Han, Mao, and Dally 2016; Guo, Yao, and Chen 2016; Dong and Yang 2019) or regularization-based pruning algorithms (Wen et al. 2016; Yu et al. 2018; He et al. 2019).

This work develops RT3D framework, including the derivation of best-suited (structured) sparsity scheme and pruning algorithm of 3D CNNs, and the design of the associated compiler-aided acceleration, for off-the-shelf mobile devices. We propose and investigate *two structured sparsity schemes* that are highly mobile acceleration friendly. The first vanilla sparsity scheme achieves sparsity by removing kernel groups in 3D CONV layers. It can achieve straightforward acceleration for on-device inference with the aid of compiler code generation, but it suffers from relatively high accuracy loss as whole kernel groups are pruned. The second, more optimized one is the *kernel group structured* (KGS) sparsity scheme. It is a more fine-grained structured sparsity that enjoys higher flexibility, and will result in a higher accuracy under the same pruning rate. Moreover, it is important to note that the KGS sparsity scheme is beyond a mere tradeoff of accuracy and mobile performance. In fact, with proper support of compiler code generation, the KGS sparsity can *achieve almost the same mobile acceleration (e.g., in frames/second) as the vanilla sparsity*, under the same pruning rate. This is owing to the delicate design of KGS sparsity to match the parallelization mechanism in compiler-assisted mobile acceleration, such that the full on-device parallelism can be exploited.

We further present *three pruning algorithms* to achieve the proposed structured sparsity schemes for 3D CNNs. The first two, i.e., the heuristic algorithm and regularization-based algorithm, are natural generalization from state-of-the-art algorithms on 2D CNN weight pruning. However, they are either greedy algorithm, or suffer from the limitation that all weights will be equally penalized even after convergence of the pruning process. Both result in potential accuracy loss. To overcome these shortcomings, we propose a novel *reweighted regularization pruning* algorithm. The basic idea is to systematically and dynamically reweight the penalties, reducing the penalties on weights with large magnitudes (which are likely to be more critical), and increasing the penalties on weights with smaller magnitudes. It possesses other advantages, such as not introducing additional hyperparameters, and being flexible for either parameter reduction or FLOPs (floating-point operations) reduction, etc.

Seamlessly integrated with above innovations, RT3D also develops the *first* end-to-end, compiler-assisted acceleration framework of 3D CNNs on both mobile CPUs and GPUs (the few prior work are limited to mobile CPUs), and also the *first* to support different structured sparsity schemes. RT3D achieves up to $29.1\times$ speedup in end-to-end inference time comparing with current mobile frameworks supporting 3D CNNs, with moderate 1%-1.5% accuracy loss, on representative CNNs (C3D, R(2+1)D, S3D). The end-to-end inference time for 16 video frames could be within 150 ms.

A brief contribution summary is: (a) sparsity schemes for 3D CNNs which are both flexible and mobile acceleration friendly, (b) highly effective pruning algorithm to achieve such sparsity schemes, (c) compiler-assisted mobile acceleration framework, and (d) for the first time, real-time performance of 3D CNNs can be achieved on off-the-shelf mobile devices using a pure software solution.

2 Related Work

2.1 Weight Pruning for 2D CNNs

The rich literature in weight pruning for 2D CNNs can be categorized into *heuristic pruning algorithms* and *regularization-based pruning algorithms*. The former starts from the early work on irregular, unstructured weight pruning where arbitrary weights can be pruned. (Han, Mao, and Dally 2016) adopts an iterative algorithm to eliminate weights with small magnitude and perform retraining to regain accuracy. (Guo, Yao, and Chen 2016) incorporates connection splicing into the pruning process to dynamically recover the pruned connections that are found to be important. Later, heuristic pruning algorithms have been generalized to the more hardware-friendly structured sparsity schemes. In (Dong and Yang 2019), Transformable Architecture Search (TAS) is adopted to realize the pruned network and knowledge is transferred from the unpruned network to the pruned version. The work (Luo, Wu, and Lin 2017) leverages a greedy algorithm to guide the pruning of the current layer with input information of the next layer. The work (Yu et al. 2018) defines a "neuron importance score" and propagates this score to conduct the weight pruning process.

Regularization-based pruning algorithms, on the other hand, are more mathematics-oriented and have the unique advantage for dealing with structured pruning problems through group Lasso regularization (Yuan and Lin 2006; Liu et al. 2018). Early work (Wen et al. 2016; He, Zhang, and Sun 2017) incorporate ℓ_1 or ℓ_2 regularization in loss function to solve filter/channel pruning problems. However, there is also one limitation of the direct application of regularization terms – all weights will be penalized equally even after pruning convergence, resulting in potential accuracy loss. A number of subsequent work are dedicated to making the regularization penalty a dynamic and "soft" term. The method in (He et al. 2018) selects filters based on ℓ_2 norm and updates the filters that have been previously pruned. (Zhang et al. 2018; Li et al. 2019) incorporate the advanced optimization solution framework ADMM (Alternating Direction Methods of Multipliers) to achieve dynamic regularization penalty, thereby improving accuracy. (He et al. 2019) proposes to adopt Geometric Median, a classic robust estimator of centrality for data in

Euclidean spaces. A common limitation of these improved versions is that the pruning rate for each layer needs to be manually set, which is difficult to derive in prior.

2.2 Mobile Acceleration Frameworks of CNNs

TVM (Chen et al. 2018), TFLite (TensorFlow 2017), Alibaba Mobile Neural Network (MNN) (Alibaba 2020) and PyTorch Mobile (PyTorch) (PyTorch 2019) are representative compiler-assisted deep learning acceleration frameworks on mobile devices. They mainly focus on end-to-end acceleration for 2D CNNs. Only MNN and PyTorch support 3D CONV on mobile CPUs (no mobile GPU support); while other popular ones (like TVM and TFLite) do not support 3D CONV computation. To the best of our knowledge, our RT3D is the **first** end-to-end deep learning acceleration framework for 3D CNNs on both mobile CPUs and GPUs. More than that, it is also the **first** to support the acceleration of various sparsity schemes of 3D CNNs. Moreover, several hardware solutions for 3D CNN acceleration have been proposed, e.g. (Hegde et al. 2018; Shen et al. 2018; Chen et al. 2019). Different from these valuable solutions that require special hardware design, RT3D employs a **pure software** solution on off-the-shelf mobile devices that is more cost-effective.

3 Structured Sparsity Schemes for 3D CNNs

This section proposes two structured sparsity schemes of 3D CNNs. We focus on the most computationally intensive convolutional (CONV) layers of 3D CNNs. Let $\mathbf{W}_l \in \mathbb{R}^{M\times N\times K_h\times K_w\times K_d}$ denote the 5-dimensional weight tensor of the l-th CONV layer of a 3D CNN, where M is the number of filters; N is the number of input channels; K_w, K_h, and K_d are the width, height, and depth, respectively, of the 3D CONV kernels. Different from the 2D CONV kernel, the 3D CONV kernel has an additional dimension on the kernel depth, making $\mathbf{W}_l$ a 5-dimensional tensor.

Figure 1 demonstrates the proposed two structured sparsity schemes for 3D CNNs: *Vanilla Structured Sparsity Scheme* and *Kernel Group Structured (KGS) Sparsity Scheme*. The weight tensor $\mathbf{W}_l$ is first partitioned into *groups of kernels* along the filter and input channel dimensions. Each kernel group consists of $g_M \times g_N$ (2×2 in Figure 1) 3D kernels. The Vanilla sparsity scheme is shown in Figure 1 (a), where whole kernel groups are determined to be pruned or not. On the other hand, our proposed KGS sparsity scheme as shown in Figure 1 (b) is that for the kernels in the same group, weights are pruned at the same locations. This is illustrated better on the right of Figure 1 (b), where 3D kernels are reshaped into vectors with $K_s = K_h \times K_w \times K_d$ weights. Consider the $g_M \times g_N$ kernels in a group, i.e., kernels $\mathbf{W}_l(m : m+g_M-1, n : n+g_N-1, :, :, :)$. Weights at the same location in these kernels i.e., $\mathbf{W}_l(m : m+g_M-1, n : n+g_N-1, h, w, d)$ are determined to be pruned or not together, where $(:, :, h, w, d)$ describes the same location (coordinate) in kernels.

The Vanilla sparsity scheme is a relatively straightforward generalization from structured sparsity schemes (Wen et al. 2016; Liu et al. 2017; Luo, Wu, and Lin 2017) for 2D CNNs. It can achieve straightforward acceleration for on-device inference with the aid of compiler code generation, but it will obviously suffer from relatively high accuracy loss as whole kernel groups are pruned. On the other hand, the proposed KGS sparsity scheme is a more fine-grained structured sparsity that enjoys higher flexibility. In fact, the Vanilla sparsity scheme is just a special case of KGS sparsity, and therefore, one can confidently state that the KGS sparsity will result in a higher accuracy under the same pruning rate, as long as effective pruning algorithm has been developed and employed.

It is important to note that the KGS sparsity scheme is beyond a mere tradeoff of accuracy and mobile performance. In fact, with proper support of compiler code generation, the KGS sparsity can *achieve almost the same mobile acceleration performance (e.g., in frames/second) as Vanilla sparsity*, under the same pruning rate. This is owing to the delicate design of KGS sparsity to match compiler-assisted mobile acceleration. For effective mobile acceleration, the whole kernel group will be transformed into matrix multiplication (with input feature map) (Chetlur et al. 2014) as shown in the reshaping step of Figure 1 (b). Accordingly, the KGS sparsity is equivalent to whole column removals in the weight matrix of a kernel group. The computation overhead in whole column removal is minor and can be mitigated by compilers, and the remaining computation is still based on full matrices (albeit smaller). A key observation is that the parallelism degree on off-the-shelf mobile devices is limited, and thus the smaller matrices of remaining weights *have enough size to fully exploit the parallelism provided by mobile devices*. As an illustrative example, suppose that the mobile device can execute 10 operations in parallel while the matrix contains 100 remaining operations. Then the reduced-size matrix can be executed in 10 iterations, achieving full parallelism. As the hardware parallelism can be fully exploited in both Vanilla and KGS schemes (if compiler overhead is negligible), the mobile acceleration performance in terms of FLOPs/second will be almost the same for both pruning schemes, and so does the frames/second performance under the same pruning rate (and FLOPs count). As a result, the proposed KGS sparsity can fully enjoy the benefit of high flexibility in terms of higher accuracy or higher pruning rate.

Please note that the $g_M \times g_N$ group size needs to be determined in Vanilla and KGS sparsity schemes, in order to achieve the maximum on-device parallelism with low computation overhead. The group size is determined offline with actual mobile testings using synthesized CNN layers. In other words, it will NOT become a hyperparameter in the pruning algorithm. $g_N = 4$ and $g_M = 4$ or 8 are preferred to match the SIMD (Single Instruction, Multiple Data) parallelism provided by current mobile CPUs and GPUs. These values are large enough to exploit the on-device parallelism and small enough to provide enough pruning flexibility and accuracy, as shall be seen in the experimental results.

4 Structured Sparsity Learning Algorithms

This section describes three pruning algorithms to achieve the proposed structured sparsity schemes for 3D CNNs. The first two are natural generalization from state-of-the-art algorithms on 2D CNN weight pruning, and the last one is specifically designed to address the limitations in the prior two. Consider a general 3D CNN consisting of L convolutional

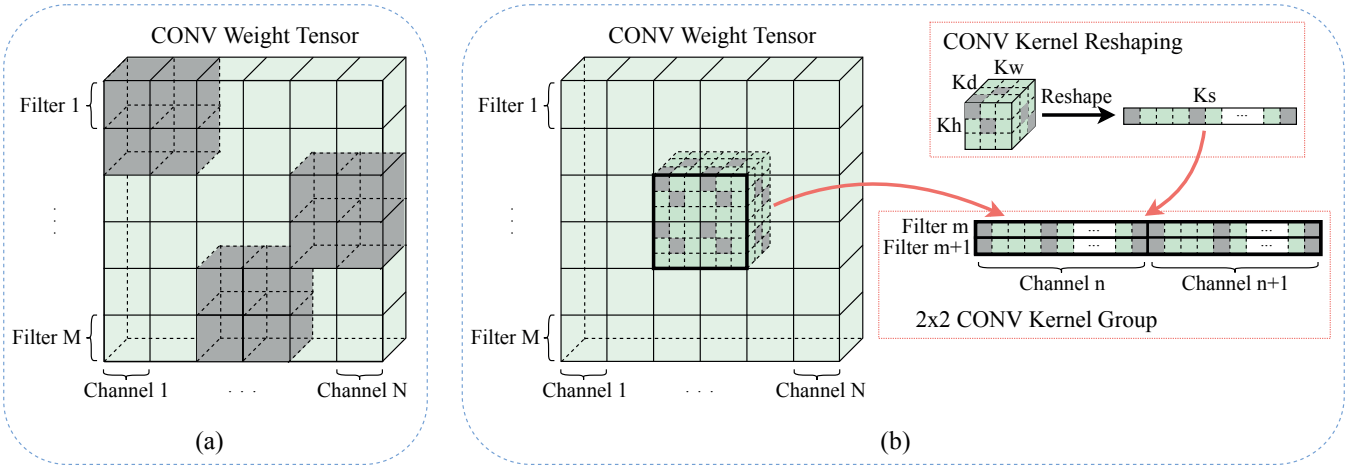

Figure 1: Proposed structured sparsity schemes: (a) The Vanilla Structured Sparsity. (b) The Kernel Group Structured (KGS) Sparsity for 3D CNNs. A CONV weight tensor is first split into multiple *kernel groups*, each consisting of $g_M \times g_N$ (2×2 in the figure) 3D kernels. Within the same kernel group, kernels are pruned at the same locations (marked by the grey entries).

(CONV) layers. Besides the l-th CONV layer weight tensor $\mathbf{W}_l$, the bias is denoted by $\mathbf{b}_l$. The loss function associated with a 3D CNN can be denoted by $F(\{\mathbf{W}_l\}_{l=1}^L, \{\mathbf{b}_l\}_{l=1}^L)$. To achieve the proposed group-wise sparsity schemes, weight tensor $\mathbf{W}_l$ is partitioned into a set of kernel groups along the dimensions of filters and channels, i.e., $\{\mathbf{W}_l^{\mathcal{G}_{p,q}}\}$, for $p \in [P]$ and $q \in [Q]$, where $P = \lceil M/g_M \rceil$, $Q = \lceil N/g_N \rceil$, and $[n]$ denotes the integer set $\{1, 2, \ldots, n\}$. Figure 2 provides an illustrative example of kernel groups.

1. Heuristic Pruning Algorithm: As discussed in Section 2.1, the prior work has investigated heuristic pruning for 2D CNNs, for both irregular and structured sparsity schemes. The prior work (Luo, Wu, and Lin 2017; Yu et al. 2018) are mostly relevant to this work as we also focus on structured sparsity. Motivated by these work, we assign a similar "neuron importance score" to each kernel group (or the same location of kernels in the group), and perform pruning on the current layer with input information of the next layer in a back propagated manner (similar procedure as (Luo, Wu, and Lin 2017)). This serves as our heuristic pruning algorithm for the proposed sparsity schemes of 3D CNNs.

2. Regularization-based Pruning Algorithm: adds an additional regularization term to the loss function to achieve the Vanilla or KGS sparsity scheme. Then, the regularization-based pruning can be formulated as

$$\underset{\{\mathbf{W}_l\},\{\mathbf{b}_l\}}{\text{minimize}} \; F\big(\{\mathbf{W}_l\}_{l=1}^L, \{\mathbf{b}_l\}_{l=1}^L\big) + \lambda \sum_{l=1}^{L} R_g\big(\mathbf{W}_l\big), \quad (1)$$

where $R_g\big(\mathbf{W}_l\big)$ is the regularization term for the Vanilla or KGS sparsity and λ is the penalty measuring its importance. Motivated by *group Lasso* (Yuan and Lin 2006), the regularization term can be defined as $R_g\big(\mathbf{W}_l\big) = \sum_{p=1}^{P}\sum_{q=1}^{Q} \left\| \mathbf{W}_l^{\mathcal{G}_{p,q}} \right\|_g$, where $\|\cdot\|_g$ denotes kernel group ℓ_p norm. We can choose from ℓ_1 norm (Liu et al. 2017), ℓ_2 norm (He et al. 2018; Li et al. 2019) or their combination for this group-wise regularization.

In more details, the regularization-based pruning can be achieved by

$$\underset{\{\mathbf{W}_l\},\{\mathbf{b}_l\}}{\text{minimize}} \; F\big(\{\mathbf{W}_l\}_{l=1}^L, \{\mathbf{b}_l\}_{l=1}^L\big) + \lambda \sum_{l=1}^{L}\sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{h=1}^{K_h}\sum_{w=1}^{K_w}\sum_{d=1}^{K_d} \left\| \mathbf{W}_l^{\mathcal{G}_{p,q}}(:,:,h,w,d) \right\|_g . \quad (2)$$

3. Reweighted Regularization Pruning Algorithm: As discussed in Section 2.1, the fixed regularization-based pruning algorithm has a limitation, – all weights will be equally penalized even after convergence of the pruning process, resulting in potential accuracy loss. We propose a novel reweighted regularization pruning algorithm to overcome this limitation. The basic idea is to systematically and dynamically reweight the penalties. Especially, we will reduce the penalties on weights with large magnitudes (which are likely to be more critical), and increase the penalties on weights with smaller magnitudes. This shall be performed in a systematic, gradual way to avoid the greedy solution which prunes a large number of weights at the early stage. Moreover, our proposed algorithm does not need to manually set the pruning rate for each layer, as a limitation in prior works based on ADMM or Geometric Median-based regularizations.

For reweighted regularization, we minimize the following objective function:

$$\underset{\{\mathbf{W}_l\},\{\mathbf{b}_l\}}{\text{minimize}} \; F\big(\{\mathbf{W}_l\}_{l=1}^L, \{\mathbf{b}_l\}_{l=1}^L\big) + \lambda \Bigg[\sum_{l=1}^{L}\sum_{p=1}^{P}\sum_{q=1}^{Q} \sum_{h=1}^{K_h}\sum_{w=1}^{K_w}\sum_{d=1}^{K_d} \left(\mathcal{P}_{l,t}^{\mathcal{G}_{p,q}} \circ \left\| \mathbf{W}_l^{\mathcal{G}_{p,q}}(:,:,h,w,d) \right\|_g \right) \Bigg], \quad (3)$$

where $\circ$ denotes element-wise multiplication. $\mathcal{P}_{l,t}^{\mathcal{G}_{p,q}}$ is the collection of penalty parameters and is updated in every iteration t to facilitate the degree of sparsity. In each iteration,

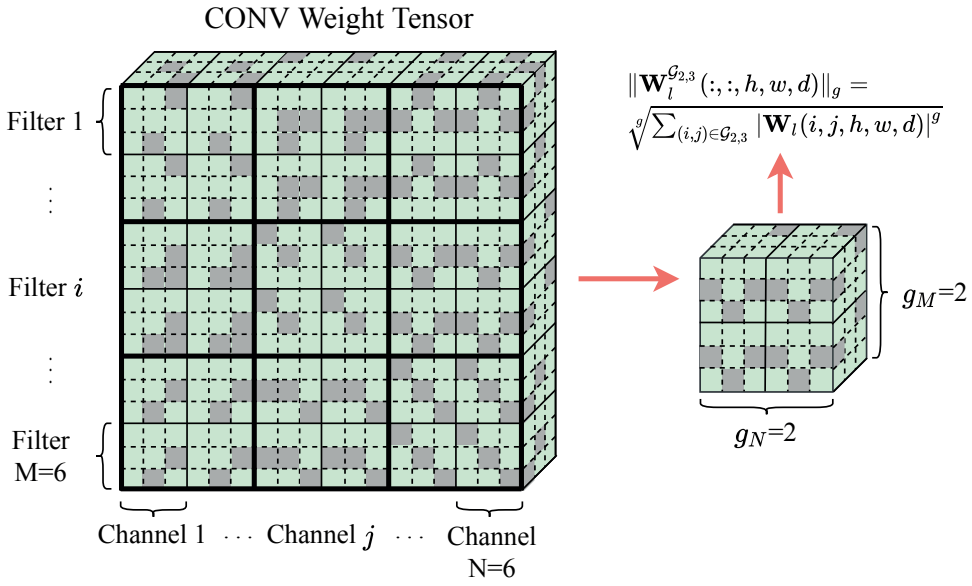

Figure 2: An example of kernel groups, each consisting of $g_M \times g_N$ (2×2) 3D kernels. Within the same kernel group, kernels are pruned at the same locations (marked by the grey entries). To achieve the same sparsity pattern for kernels in the same group, group lasso is calculated as $\left\|\mathbf{W}_l^{\mathcal{G}_{p,q}}(:,:,h,w,d)\right\|_g = \sqrt[g]{\sum_{(i,j)\in\mathcal{G}_{p,q}} |\mathbf{W}_l(:,:,h,w,d)|^g}$.

the instance of $\mathbf{W}_l^{\mathcal{G}_{p,q}}$ is denoted by $\mathbf{W}_{l,t}^{\mathcal{G}_{p,q}}$ and we update $\mathcal{P}_{l,t}^{\mathcal{G}_{p,q}}$ by setting

$$\mathcal{P}_{l,(t+1)}^{\mathcal{G}_{p,q}} = \frac{1}{\left\|\mathbf{W}_{l,t}^{\mathcal{G}_{p,q}}(:,:,h,w,d)\right\|_g^2 + \epsilon},$$

where ϵ is a small positive number avoiding the zero denominator. The reweighted regularization process updates penalty parameters based on the current weight values, will not incur extra hyperparameters, and has a fast convergence rate as analyzed in (Candes, Wakin, and Boyd 2008). After 3∼4 iterations, we will prune the weights that converge to zero, and perform a slight retraining on the non-zero weights (with a few epochs) to regain accuracy.

While overcoming the limitation in fixed regularization-based algorithms, the advantage and flexibility in such algorithms will be preserved. There is only one λ as the major hyperparameter, without the need of manually deciding per-layer pruning rate. Also similar to fixed regularization algorithms, we can multiply the per-layer FLOPs value to each layer l in the above optimization function. In this way we can target at the overall FLOPs reduction, which is more relevant to the actual acceleration. In the experiments, we set the FLOPs reduction as the optimization target, and report the corresponding FLOPs reduction rates and actually measured mobile accelerations.

5 Experimental Results

5.1 Evaluation on Sparsity Schemes and Pruning Algorithms

Experimental Setup. We test the proposed two structured sparsity schemes i.e., Vanilla and KGS sparsity and three pruning algorithms on 3D CNN models (including one (2+1)D CNN): C3D (Tran et al. 2015), R(2+1)D (Tran et al. 2018), and S3D (Xie et al. 2018). Besides the two proposed sparsity schemes, a filter sparsity scheme is also implemented, where the filters may be pruned as a whole, and which is a direct generalization of the filter pruning of 2D CNNs. The models are all pretrained on the Kinetics dataset (Carreira and Zisserman 2017) and transferred onto the UCF101 (Soomro, Zamir, and Shah 2012) and HMDB51 (Kuehne et al. 2011) datasets as the pretrained dense models. The hyperparameter settings are the same for all pruning algorithms and sparsity schemes for fair comparisons. The batch size is fixed to 32, and the video clip length is 16 frames. The initial learning rate is 5e−3 when training the dense model, and is reduced to 2e−4 in the weight pruning and retraining for stability. The learning rate is fixed in the pruning process, while adjusted in retraining with a scheduler following the cosine function. For different types of sparsity schemes and pruning algorithms, the total number of epochs is fixed to 240 epochs.[3] For the pruning of all the models, we have used the best combination of ℓ_1 and ℓ_2 norms in the regularization term. The penalty factor λ is set to 5e−4. The pruning and retraining processes are carried out with eight NVIDIA GeForce GTX 1080 Ti GPUs

[3] Although the reweighted pruning algorithm is iterative, its latter iterations require significantly fewer epochs. Thus we can set the total epochs the same for different algorithms.

Model	Pruning Algorithm	Sparsity Scheme	Overall FLOPs after Pruning	Pruning Rate of FLOPs	Base Top-1 Accuracy	Pruning Top-1 Accuracy
C3D (299MB)	Heuristic	Filter	15.2G	2.6×	81.6%	78.6%
		Vanilla	15.2G	2.6×		78.8%
		KGS	15.2G	2.6×		79.0%
		KGS	10.8G	3.6×		78.5%
	Regularization	Filter	15.2G	2.6×	81.6%	78.8%
		Vanilla	15.2G	2.6×		79.0%
		KGS	15.2G	2.6×		79.6%
		KGS	10.8G	3.6×		79.3%
	Reweighted Regularization	Filter	15.2G	2.6×	81.6%	79.3%
		Vanilla	15.2G	2.6×		79.7%
		KGS	**15.2G**	**2.6**×		**80.5**%
		KGS	**10.8G**	**3.6**×		**80.2**%
R(2+1)D (120MB)	Heuristic	Filter	15.9G	2.6×	94.0%	89.0%
		Vanilla	15.9G	2.6×		89.4%
		KGS	15.9G	2.6×		90.4%
		KGS	12.7G	3.2×		89.9%
	Regularization	Filter	15.9G	2.6×	94.0%	89.8%
		Vanilla	15.9G	2.6×		90.8%
		KGS	15.9G	2.6×		91.7%
		KGS	12.7G	3.2×		91.3%
	Reweighted Regularization	Filter	15.9G	2.6×	94.0%	90.5%
		Vanilla	15.9G	2.6×		91.7%
		KGS	**15.9G**	**2.6**×		**92.5**%
		KGS	**12.7G**	**3.2**×		**92.0**%

Table 1: 3D CNN pruning results on the UCF101 dataset.

on Ubuntu operating system and the PyTorch 1.3 framework with CUDA 10.1. The total required memory is up to 38.8GB.

Results. The pruning results on C3D, R(2+1)D models on UCF101 dataset with various pruning algorithms and sparsity schemes are provided in Table 1. For each pruning algorithm, the three sparsity schemes are compared under the same pruning rate (FLOPs reduction on the overall model), and KGS results of two pruning configurations are compared. As can be observed in the table, the KGS sparisity scheme consistently outperforms the vanilla sparsity, and these two schemes both perform better than filter pruning. The reweighted regularization algorithm consistently outperforms the other two pruning algorithms. The advantages of KGS sparsity and reweighted regularization are stated in Section 3 and Section 4. With reweighted regularization and KGS sparsity scheme, both C3D and R(2+1)D could achieve only 1%∼1.5% accuracy loss under pruning rate of 2.6×.

5.2 Evaluation on Mobile Acceleration Performance

Mobile Acceleration Framework Implementation. We design and implement an end-to-end, compiler-assisted CNN acceleration framework that supports 3D CNNs. Without any pruning-related optimizations, RT3D is already faster than state-of-the-art CNN execution frameworks (such as MNN and PyTorch Mobile) on mobile CPUs, because we include more advanced optimizations like fine-tuned high-efficient SIMD (Single Instruction, Multiple Data) execution, fine-tuned weight layout organization, etc. Our framework is also the first to support 3D CNN executions on mobile GPUs. It is general, supporting both 2D and 3D CNNs. Comparing to other popular CNN acceleration frameworks that support 2D CONV (like TVM and MNN) on standard 2D benchmarks like VGG-Net, ResNet, MobileNet-V2, etc., our developed framework also yields consistently better performance.

Moreover, RT3D is also the first to support various sparsity schemes, including Filter, and proposed Vanilla and KGS sparsity. Based on the sparsity scheme, it employs a compiler-based automatic code generation approach to reorganize the model weights, regularize the computations, tune the computation configuration, and generate the optimized model inference codes. Our framework can automatically generate both optimized CPU (vectorized C++) and GPU (OpenCL) codes to support both dense and sparse 3D CNN executions.

Test-bed and Evaluation Setup. The evaluations are conducted on a Samsung Galaxy S20 cellphone with the latest Qualcomm Snapdragon 865 platform consisting of a Qualcomm Kryo 585 Octa-core CPU and a Qualcomm Adreno 650 GPU. All experiments run 50 times with 8 threads on mobile CPU, and all pipelines on mobile GPU. Because different runs do not vary severely, only the average inference execution time is reported for readability. All models are tuned to their best configurations, e.g., with computational graph optimizations, the best tiling size, unrolling size, etc. 32-bit

Framework	MNN	PyTorch	RT3D (Dense)				RT3D (Sparse)			
Device	CPU (ms)	CPU (ms)	CPU (ms)	Speedup	GPU (ms)	Speedup	CPU (ms)	Speedup	GPU (ms)	Speedup
C3D	948	2544	902	2.8×	488	5.2×	**357**	**7.1**×	**142**	**17.9**×
R(2+1)D	-	4104	1074	3.8×	513	8.0×	**391**	**10.5**×	**141**	**29.1**×
S3D	-	6617	1139	5.8×	565	11.7×	**611**	**10.8**×	**293**	**22.6**×

Table 2: Inference latency comparison of RT3D, MNN, and PyTorch on mobile CPU and GPU. MNN does not support R(2+1)D and S3D yet. For RT3D (Sparse), all models are pruned by reweighted regularization algorithm with KGS sparsity. The pruning rate (in FLOPs) is $3.6\times$ for C3D, $3.2\times$ for R(2+1)D, and $2.1\times$ for S3D, and the accuracy is 80.2%, 92.0%, and 90.2%, respectively.

Model	Sparsity Scheme	Base Top-1 Accuracy	Pruning Top-1 Accuracy	FLOPs after Pruning	Pruning Rate of FLOPs	Latency (ms) CPU	Latency (ms) GPU
C3D	Vanilla	81.6%	80.0%	16.4G	2.4×	525	236
	KGS			**9.7G**	**4.0**×	**329**	**134**
R(2+1)D	Vanilla	94.0%	91.8%	15.5G	2.5×	523	225
	KGS			**10.2G**	**4.0**×	**360**	**127**

Table 3: Comparison between Vanilla and KGS sparsity schemes: pruning rate, and inference latency with the same pruning Top-1 accuracy on the UCF101 dataset. Reweighted regularization pruning is applied for all models.

floating point is applied for CPU runs, and 16-bit floating point is used for GPU runs. This is the same for both baseline mobile acceleration frameworks and our RT3D framework for a fair comparison, as quantization is not supported by baseline frameworks.

Mobile Acceleration Results. We next evaluate RT3D by comparing it with MNN (Alibaba 2020) and PyTorch Mobile (PyTorch) (PyTorch 2019).[4] Table 2 compares the end-to-end 3D CNN inference time (latency). RT3D supports both dense (original) and sparse 3D CNNs on both mobile CPU and mobile GPU, PyTorch supports dense models on CPU only, and MNN supports dense C3D on CPU only. For sparse models, RT3D uses pruned models by reweighted regularization pruning algorithms with KGS sparsity with the pruning rate of $3.6\times$ for C3D, $3.2\times$ for R(2+1)D, and $2.1\times$ for S3D, and the accuracy of 80.2%, 92.0%, and 90.2%[5], respectively. In the table, the RT3D speedups are compared with PyTorch. RT3D outperforms MNN and PyTorch on mobile CPU for all cases. RT3D on mobile GPU performs even better than on CPU. For example, for C3D, the fully optimized RT3D (Sparse) outperforms the CPU version of PyTorch and MNN with the speedup of $7.1\times$ and $2.7\times$ on CPU, and $17.9\times$ and $6.7\times$ on GPU, respectively. Notably, on mobile GPU, the fully optimized RT3D can infer 16 frames by using C3D, R(2+1)D, and S3D within 142 ms, 141 ms, and 293 ms, respectively, achieving real-time execution (say 30 frames per second) of 3D CNNs on mobile devices.

Importantly, although RT3D's dense implementations have already been fully optimized, our sparse implementations especially on mobile GPU can fully transform the pruning rate of FLOPs into inference latency speedup. For example, on C3D, from RT3D (dense) to RT3D (sparse) on GPU, the improvement on inference latency is $3.43\times$, while the pruning rate of the sparse model is $3.6\times$. This validates the statement in Section 3 that the proposed KGS sparsity scheme can exploit the parallelism degree on device. Moreover, 3D CONV is memory-intensive, bounded by both memory bandwidth and latency (which is more severe on mobile GPU due to its even limited cache capacity), and our pruning/compilation co-design is able to mitigate this issue with incurring negligible overhead. Our cache access count results validate this.

Ablation Study. We also compare two sparsity schemes, Vanilla and KGS in terms of pruning rate and inference latency on mobiles by controlling the same pruning top-1 accuracy (as shown in Table 3). It shows that KGS scheme achieves both higher pruning rate (in FLOPs) and lower inference latency under the same pruning accuracy on both C3D and R(2+1)D due to KGS's high flexibility and seamless match with compiler-level optimizations.

6 Conclusion

This paper presents RT3D, a mobile acceleration framework for 3D CNNs that includes two novel, mobile-friendly structured sparsity schemes (Vanilla and KGS) and best-suited pruning algorithms, that can achieve low inference latency and high accuracy, simultaneously. Based on them, RT3D employs a compiler-assisted code generation framework to transform pruning benefits to performance gains. The evaluation results show that RT3D beats two state-of-the-art acceleration frameworks with speedup up to $29.1\times$. RT3D can infer 16 video frames within 150 ms, for the first time, achieving real-time inference of 3D CNNs on off-the-shelf mobile devices with a pure software solution.

[4] Other popular mobile CNN acceleration frameworks like TVM and TFLite do not support 3D CNNs.

[5] The base accuracy of S3D is 90.6%.

Ethics Statement

As a pure software solution, RT3D is the first to achieve real-time execution of 3D CNNs on mobile devices without notable accuracy loss–which can only be achieved by special (and more expensive) hardware solutions previously. This research will significantly encourage the general research of deep learning acceleration with software-based techniques while reducing the demand for some hardware-based accelerations. RT3D will enable many machine learning applications of behavior/activity detection on mobile platforms that have to run on the cloud previously.

The ethical aspects and future societal consequences of this research are highly application-dependent. This work has the following potential positive impact in society: First, because machine learning applications can run on the mobile (edge) side only without transmitting user data to the cloud server, data privacy is significantly enhanced, thus these applications can run in a more private environment. Second, this work may also significantly broaden the usage of machine learning in many other domains ranging from medical science, biology to geoscience and environmental science, e.g., combining motion sensors and high-precision 3D CNN inferences to support short-latency motion recognition can enable a real-time Parkinson treatment. At the same time, this work may have some negative consequences: due to the low-cost and easy-accessible nature of machine learning on mobile, this work has the potential of increasing the possibility of misusing machine learning techniques. Furthermore, we should be cautious of the result of the failure of the system which could cause wrong decision making, thus jeopardizing the safety of the public and individuals. In addition, all experiments in our work are based on the public dataset and our task/method does not leverage biases in the data.

Acknowledgement

This material is based upon work supported by the National Science Foundation (NSF) under Grants CCF-1937500, CNS-1932351, Army Research Office Young Investigator Program 76598CSYIP, and Jeffress Trust Awards in Interdisciplinary Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, or Thomas F. and Kate Miller Jeffress Memorial Trust.

References

Alibaba. 2020. https://github.com/alibaba/MNN.

Candes, E. J.; Wakin, M. B.; and Boyd, S. P. 2008. Enhancing sparsity by reweighted $\ell 1$ minimization. *Journal of Fourier analysis and applications* 14(5-6): 877–905.

Carreira, J.; and Zisserman, A. 2017. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 6299–6308.

Chen, H.; Song, M.; Zhao, J.; Dai, Y.; and Li, T. 2019. 3D-based video recognition acceleration by leveraging temporal locality. In *Proceedings of the 46th International Symposium on Computer Architecture*, 79–90.

Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; Guestrin, C.; and Krishnamurthy, A. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 578–594.

Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; and Shelhamer, E. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* .

Dong, X.; and Yang, Y. 2019. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 759–770.

Guo, Y.; Yao, A.; and Chen, Y. 2016. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems (NeurIPS)*, 1379–1387.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*.

Han, S.; Shen, H.; Philipose, M.; Agarwal, S.; Wolman, A.; and Krishnamurthy, A. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 123–136. ACM.

He, Y.; Kang, G.; Dong, X.; Fu, Y.; and Yang, Y. 2018. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4340–4349.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 1389–1397.

Hegde, K.; Agrawal, R.; Yao, Y.; and Fletcher, C. W. 2018. Morph: Flexible acceleration for 3D CNN-based video understanding. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 933–946. IEEE.

Huynh, L. N.; Lee, Y.; and Balan, R. K. 2017. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 82–95. ACM.

Ji, S.; Xu, W.; Yang, M.; and Yu, K. 2012. 3D convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence (T PAMI)* 35(1): 221–231.

Köpüklü, O.; Kose, N.; Gunduz, A.; and Rigoll, G. 2019. Resource efficient 3d convolutional neural networks. In *2019*

IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), 1910–1919. IEEE.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NeurIPS)*, 1097–1105.

Kuehne, H.; Jhuang, H.; Garrote, E.; Poggio, T.; and Serre, T. 2011. HMDB: a large video database for human motion recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2556–2563. IEEE.

Li, T.; Wu, B.; Yang, Y.; Fan, Y.; Zhang, Y.; and Liu, W. 2019. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3977–3986.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2736–2744.

Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; and Darrell, T. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* .

Luo, J.-H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, 5058–5066.

PyTorch. 2019. https://pytorch.org/mobile/home.

Qiu, Z.; Yao, T.; and Mei, T. 2017. Learning spatio-temporal representation with pseudo-3d residual networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 5533–5541.

Shen, J.; Huang, Y.; Wang, Z.; Qiao, Y.; Wen, M.; and Zhang, C. 2018. Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 97–106.

Soomro, K.; Zamir, A. R.; and Shah, M. 2012. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402* .

TensorFlow. 2017. https://www.tensorflow.org/mobile/tflite/.

Tran, D.; Bourdev, L.; Fergus, R.; Torresani, L.; and Paluri, M. 2015. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 4489–4497.

Tran, D.; Wang, H.; Torresani, L.; Ray, J.; LeCun, Y.; and Paluri, M. 2018. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 6450–6459.

Wang, Z.; Lan, Q.; He, H.; and Zhang, C. 2017. Winograd algorithm for 3D convolution neural networks. In *International Conference on Artificial Neural Networks (ICANN)*, 609–616. Springer.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems (NeurIPS)*, 2074–2082.

Xie, S.; Sun, C.; Huang, J.; Tu, Z.; and Murphy, K. 2018. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 305–321.

Yao, S.; Hu, S.; Zhao, Y.; Zhang, A.; and Abdelzaher, T. 2017. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*, 351–360.

Yu, R.; Li, A.; Chen, C.-F.; Lai, J.-H.; Morariu, V. I.; Han, X.; Gao, M.; Lin, C.-Y.; and Davis, L. S. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 9194–9203.

Yuan, M.; and Lin, Y. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68(1): 49–67.

Zhang, T.; Ye, S.; Zhang, Y.; Wang, Y.; and Fardad, M. 2018. Systematic Weight Pruning of DNNs using Alternating Direction Method of Multipliers. *arXiv preprint arXiv:1802.05747* .

Zhuang, Z.; Tan, M.; Zhuang, B.; Liu, J.; Guo, Y.; Wu, Q.; Huang, J.; and Zhu, J. 2018. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 875–886.