

Real-Time Mobile Acceleration of DNNs: From Computer Vision to Medical Applications

Hongjia Li*
Northeastern University
Boston, USA
li.hongjia@northeastern.edu

Yuxuan Cai
Northeastern University
Boston, USA
cai.yuxu@northeastern.edu

Bin Ren
College of William and Mary
Williamsburg, Virginia
bren@cs.wm.edu

Geng Yuan*
Northeastern University
Boston, USA
yuan.geng@northeastern.edu

Mengshu Sun
Northeastern University
Boston, USA
sun.meng@northeastern.edu

Xue Lin
Northeastern University
Boston, USA
xue.lin@northeastern.edu

Wei Niu
College of William and Mary
Williamsburg, Virginia
wniu@email.wm.edu

Zhengang Li
Northeastern University
Boston, USA
li.zhen@northeastern.edu

Yanzhi Wang
Northeastern University
Boston, USA
yanz.wang@northeastern.edu

ABSTRACT

With the growth of mobile vision applications, there is a growing need to break through the current performance limitation of mobile platforms, especially for computationally intensive applications, such as object detection, action recognition, and medical diagnosis. To achieve this goal, we present our unified real-time mobile DNN inference acceleration framework, seamlessly integrating hardware-friendly, structured model compression with mobile-targeted compiler optimizations. We aim at an unprecedented, real-time performance of such large-scale neural network inference on mobile devices. A fine-grained block-based pruning scheme is proposed to be universally applicable to all types of DNN layers, such as convolutional layers with different kernel sizes and fully connected layers. Moreover, it is also successfully extended to 3D convolutions. With the assist of our compiler optimizations, the fine-grained block-based sparsity is fully utilized to achieve high model accuracy and high hardware acceleration simultaneously. To validate our framework, three representative fields of applications are implemented and demonstrated, object detection, activity detection, and medical diagnosis. All applications achieve real-time inference using an off-the-shelf smartphone, outperforming the representative mobile DNN inference acceleration frameworks by up to $6.7\times$ in speed. The demonstrations of these applications can be found in the following link: <https://bit.ly/39IWpYu>.

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-7999-1/21/01...\$15.00

<https://doi.org/10.1145/3394885.3431627>

CCS CONCEPTS

- **Computing methodologies** → **Computer vision tasks;**
- **Human-centered computing** → **Mobile devices.**

KEYWORDS

computer vision, real-time, mobile acceleration

ACM Reference Format:

Hongjia Li, Geng Yuan, Wei Niu, Yuxuan Cai, Mengshu Sun, Zhengang Li, Bin Ren, Xue Lin, and Yanzhi Wang. 2021. Real-Time Mobile Acceleration of DNNs: From Computer Vision to Medical Applications. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21)*, January 18–21, 2021, Tokyo, Japan. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3394885.3431627>

1 INTRODUCTION

With the breakthrough of Deep Neural Networks (DNNs) in the past decade, neural network based computer vision has made huge progress, achieving exceptional performance. Tasks such as object detection, activity detection, and medical diagnosis are deployed in a wide range of applications, including autonomous driving, robot vision and training, human-computer interaction, and augmented reality [2, 25]. To increase the demand of application accuracy, DNN models are tuned to large scales by adding more parameters and layers. Meanwhile, mobile devices are rapidly becoming the central computer and carrier for deep learning tasks [27]. However, real-time execution has been limited due to the computation/storage resource constraints on mobile devices.

To address this issue, efforts has been devoted to reduce the computation and memory intensity of DNN models, in which model compression techniques are considering as one of the most effective approaches to reach the goal while maintaining high accuracy [8, 14, 24, 28]. Among the model compression techniques, the most trendy way leads to weight pruning [8, 14, 24]. By exploring and eliminating the redundancy in weights, DNN models are able to downscale with sparse weights, which requires fewer memory references and power consumption during inference. Weight pruning

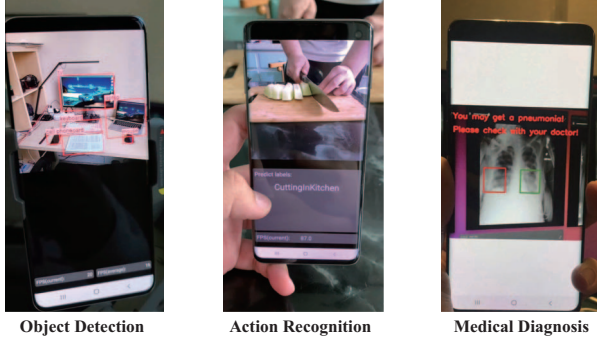


Figure 1: Illustration of applications under our mobile acceleration of DNNs framework.

schemes can be divided into three categories: unstructured pruning, structured pruning, and pattern-based pruning. To be specific, unstructured pruning searches and reduces weights in an irregular manner, which leads to incompatibility with hardware accelerations [8, 11]. More hardware-friendly, structured pruning aims at maintaining a full matrix format after weight pruning. However, pruning whole filters/channels bring notable accuracy degradation due to the coarse-grained nature [14]. To overcome the shortcomings of prior works, pattern-based pruning is recently proposed to incorporate fine-grained unstructured pruning in a hardware aware fashion [13, 16]. However, the main limitation is that it can only be applied to convolutional (CONV) layers with 3×3 kernels.

Despite promising results prior weight pruning works achieved, there exist limitations when applying them to deep learning tasks. First, only a few certain layer types can be pruned using state-of-art schemes. Not all the layers in any computer vision models can meet the requirement. Second, most of the weight pruning works focus on traditional 2D DNNs in the image domain. However, 3D DNNs have been proposed for video domain tasks such as video classification, and action recognition/detection [18]. These issues lead to extra time-consuming works when implementing real-time mobile applications on different deep learning tasks.

To overcome these shortcomings, we present our unified real-time mobile acceleration of DNNs framework, seamlessly integrating hardware-friendly, structured model compression with mobile-targeted compiler optimization. The goal of our framework is to provide an unprecedented, real-time performance of such large-scale neural network inference using simply off-the-shelf mobile devices. We propose a fine-grained block-based pruning scheme, which can be universally applicable to all types of DNN layers, such as CONV layers with different kernel sizes and fully connected layers. Moreover, it is also successfully extended to 3D convolutions. Different weight pruning schemes, such as unstructured pruning, filter/column pruning, and our block-based pruning, are analyzed and compared given the specific deep learning problems. To accommodate our proposed block-based pruning properties, unique compiler optimizations are adopted to achieve high inference accuracy and high hardware acceleration simultaneously. To validate our framework, three representative fields of applications are implemented and demonstrated, object detection, activity detection, and

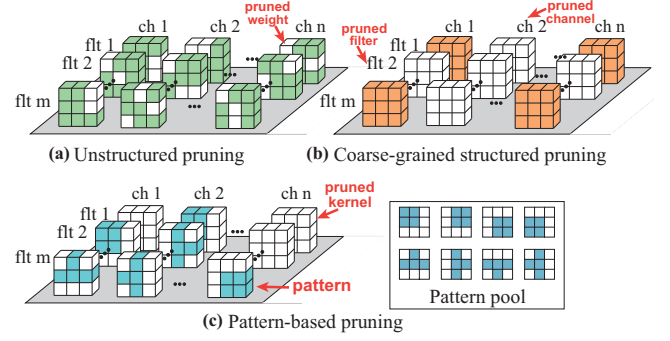


Figure 2: Illustration of (a) unstructured pruning, (b) coarse-grained structured pruning, and (c) pattern-based pruning.

medical diagnosis. All applications can achieve real-time inference on mobile devices, outperforming the current mobile acceleration framework by up to $6.7\times$ in speed.

2 BACKGROUND AND RELATED WORKS

2.1 DNN model pruning

The three main categories in weight pruning schemes are shown in Figure 2: unstructured pruning, coarse-grained structured pruning, and pattern-based pruning.

Unstructured pruning [8] is a fine-grained pruning scheme, which eliminates weights at arbitrary locations, therefore showing the advantage in maintaining the accuracy of the pruned model. However, the irregular sparse manner of pruned weights results in performance limitation when deploying on hardware. Extra indices are required to track the non-zero weights, which is harmful for the acceleration for DNN inference [12].

Coarse-grained Structured pruning [9, 24] aims at CONV layers and maintaining the full matrix format after pruning. It can be divided into filter pruning and channel pruning. In the weight matrix format representation as shown in Figure 2 (a), filter pruning corresponds to reducing rows of the weight matrix. Accordingly, channel pruning corresponds to reducing multiple consecutive columns. The full matrix weight manner achieves facilitating hardware acceleration. However, accuracy degradation is observed due to its coarse-grain nature.

Pattern-based pruning [13, 16] has been recently proposed to overcoming the shortcomings of previous works. As shown in Figure 2 (c), pattern-based pruning reveals the sparsity of DNN models at the intra-kernel level. More specifically, it prunes the weights inside of each convolutional kernel, and the remaining weights will form certain patterns. It can also remove the entire redundant kernel to further increase the compression rate. With the assist of compiler code generation, pattern-based pruning achieves both high accuracy and hardware parallelism. However, it is designed only for 3×3 CONV layers and is not applicable to other types of layers. The key difficulty in generalizing pattern-based pruning to layer sizes beyond 3×3 is that the larger kernel size results in a larger number of pattern types and incurs notable computation overhead in compiler-generated executable codes. This significantly restricts the application scenarios of pattern-based pruning.

2.2 Compiler-based DNN Frameworks on Mobile

Since mobile devices become a trendy platform for deep learning applications, research from both industry and academia has been investigated to accelerate the implementation. Various DNN execution frameworks are proposed, including representative works TensorFlow-Lite (TFLite) [21], Alibaba MNN (MNN) [15], and TVM [6]. These three DNN execution frameworks all achieve state-of-art performance on the end-to-end deployment with high execution efficiency. Optimizations are adopted in these works, including varied computation graph optimizations, tensor optimizations, half-float support, advanced parameters auto-tuning, etc. However, sparse DNN models are not supported in either of these frameworks, which is the essential drawback to obstruct the real-time DNN inference on mobile devices.

To address these issues, frameworks based on the fine-grained pattern-based pruning such as PatDNN [16] and PCONV [13] are proposed, adopting compiler-based optimizations to provide accelerated deployment for sparse DNN models. The main limitation of these frameworks is that only square and small convolution kernels used in 3×3 CONV layers are accelerated. On the other hand, larger kernel sizes such as 5×5 , 7×7 can lead to huge code execution overhead. In addition, FC layers and 1×1 CONV layers are not supported as mentioned above.

3 PROPOSED FRAMEWORK

In this section, our real-time mobile acceleration of DNNs framework is presented, seamlessly integrating hardware-friendly, structured model compression with mobile-targeted compiler optimization. The goal of our framework is to achieve a unified, unprecedented, real-time performance of large-scale neural network inference using simply off-the-shelf mobile devices. The framework is introduced from both compression and compilation aspects.

3.1 Compression: A Unified Fine-grained Block-based Pruning Scheme

As mentioned above, the existing pruning schemes have their individual shortcomings, such as limited hardware acceleration, low accuracy, limited to a certain type of layers. However, pattern-based pruning inspires a new optimization dimension that fine-grained pruning scheme delivers promising performance with the assist of compiler code generation. Thus, we propose a fine-grained block-based pruning that leverages an intermediate sparsity granularity between unstructured pruning and coarse-grained structured pruning while achieving high accuracy and hardware parallelism simultaneously. Most importantly, our block-based pruning is general to CONV layers with different kernel sizes (1×1 , 3×3 , 5×5 , and larger) and fully connected (FC) layers. Moreover, it can also be extended from 2D DNN models to 3D DNN models.

As shown in Figure 3 (a), the weights in a DNN layer are divided into a number of equal-sized blocks, each block contains the weights from n channels of m filters, and the block size is denoted as $m \times n$ (e.g., 4×4 , 16×32 , 64×128). For 2D CONV layers, in each block, two dimensions of pruning can be conducted in our block-based pruning. First, we can prune the weights at the same location among different kernels within each channel, different locations are allowed to

be pruned among different channels. Second, we can prune the entire filter among different channels. The pruning rate can be different among different blocks. If we convert the weight tensor to weight matrix format that is used in GEneral Matrix Multiplications (GEMMs) [7]. Then, the block-based pruning can be considered as pruning the columns or rows of weights within each block. For FC layers, it is similar to the weight matrix format of the 2D CONV layer. The number of rows and columns of the weight matrix equals the number of input neurons and out neurons, respectively. The rows and columns of weights within each block can be pruned.

The **high accuracy** is ensured by the fine-grained property of block-based pruning. Cause it provides higher flexibility when searching the pruned model structure compared to coarse-grained structured pruning (which prunes entire rows/columns in weight matrix), where leads to higher accuracy.

The **high hardware inference performance** is attributed to the proposed appropriate degree of structural regularity, which can be exploited by compiler-level optimization to fully enjoy the benefit of high (or even maximum) hardware parallelism. Typically, the number of weights in a DNN layer is very large. Even when we divide the weights into blocks, the computation required by each block could still be sufficient to saturate the hardware computing resources and achieve a high degree of parallelism, especially on the resource-limited mobile devices. It is worth noting that the unstructured pruning and coarse-grained structured pruning can be considered as two special cases of block-based pruning, where the block size is 1 column $\times$ 1 row and the whole weight matrix, respectively. Thus, the block-size trade-off the accuracy and hardware parallelism (acceleration), which is usually depending on the application requirements.

For 3D CONV layers, since it is more computation-intensive compared to 2D CONV layers, a more strict constraint of our block-based pruning is designed to facilitate the hardware parallelism and hence a faster inference speed. As shown in Figure 3 (b), we prune the same locations among all 3D kernels within a block. Such a pruning constraint can better leverage the hardware parallelism from both memory and computation perspectives. First, during convolution computation, all filters in each layer share the same input. Since the same locations are pruned among all the kernels, these filters will skip reading the same input data, thus mitigating the memory pressure among the threads processing these filters. Second, the restriction of pruning identical locations across channels within a block ensures that all of these channels share the same computation pattern (indices), thus eliminating the computation divergence among the threads processing the channels within each block.

3.2 Unified Regularization Pruning Algorithm

In the prior works, the two popular methods for weight pruning algorithm are group Lasso regularization [9, 24] or Alternating Direction Methods of Multipliers (ADMM) [28]. However, these two methods suffer from either potential accuracy loss or the requirement of manual compression rate tuning work. Therefore, in our framework, the reweighted group Lasso [4] method is adopted, which is to systematically and dynamically reweight the penalties. Specifically, the reweighted method reduces the penalties on

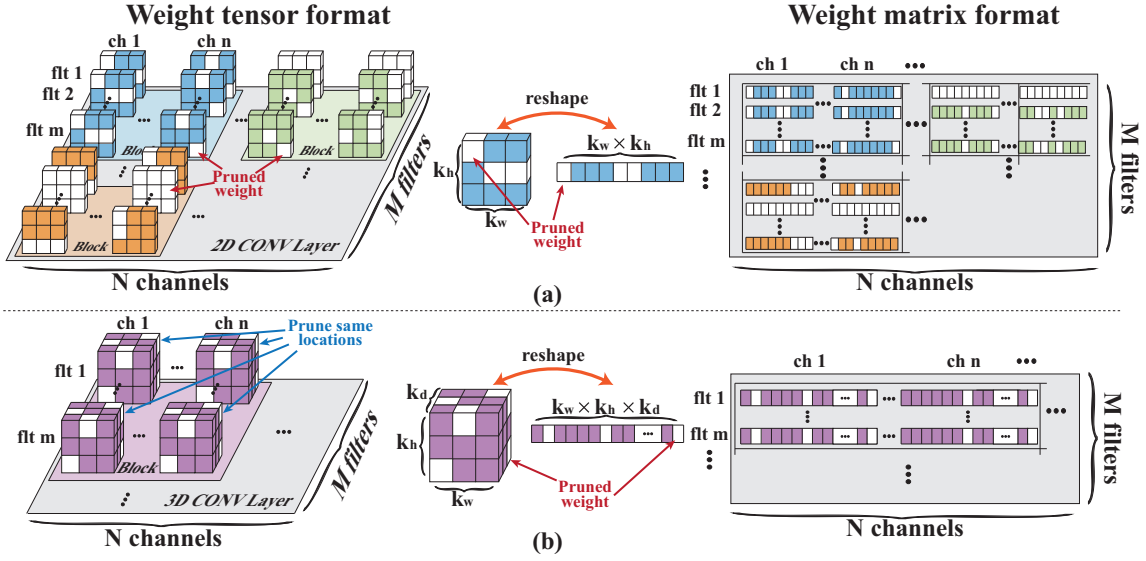


Figure 3: Example of block-based pruning for (a) 2D CONV layer, and (b) 3D CONV layer.

weights with larger magnitudes, and increases the penalties on weights with smaller magnitudes.

For **2D DNNs**: The 4-D weight tensor of i -th CONV layer of DNN is denoted by $\mathbf{W}_i \in \mathbb{R}^{M \times N \times K_h \times K_w}$, where M denotes the number of filters; N denotes the number of input channels; K_w and K_h are the width and height kernels of i -th layer. Therefore, the general reweighted pruning problem is represented as

$$\underset{\mathbf{W}, \mathbf{b}}{\text{minimize}} \quad f(\mathbf{W}; \mathbf{b}) + \lambda \sum_{i=1}^N R(\alpha_i^{(t)}, \mathbf{W}_i), \quad (1)$$

where $f(\mathbf{W}; \mathbf{b})$ is the loss function of DNN. $R(\cdot)$ denotes the regularization term used to generate model sparsity and the hyperparameter λ controls the trade-off between accuracy and sparsity. $\alpha_i^{(t)}$ represents the collection of penalty values applied on the weights $\mathbf{W}_i$ for layer i .

Applying our fine-grained block-based pruning, each $\mathbf{W}_i$ is divided into K blocks with the same size $g_m \times g_n$, namely, $\mathbf{W}_i = [\mathbf{W}_{i1}, \mathbf{W}_{i2}, \dots, \mathbf{W}_{iK}]$, where $\mathbf{W}_{ij} \in \mathbb{R}^{g_m \times g_n}$. Therefore, the regularization term is

$$R(\alpha_i^{(t)}, \mathbf{W}_i) = \sum_{j=1}^K \sum_{h=1}^{g_m} \sum_{w=1}^{g_n} \|\alpha_{ij}^{(t)} \odot [\mathbf{W}_{ij}]_{h,w}\|_F^2, \quad (2)$$

where $\alpha_{ij}^{(t)}$ is updated by $\alpha_{ij}^{(t)} = \frac{1}{\|[\mathbf{W}_{ij}]_{h,w}^t\|_F^2 + \epsilon}$.

For **3D DNNs**: Let $\mathbf{W}_i \in \mathbb{R}^{M \times N \times K_d \times K_h \times K_w}$ denotes the 5D weight tensor, in which an additional dimension, the depth of kernel K_d is considered. Under our block-based pruning, $\mathbf{W}_i$ is partitioned into a set of kernel groups along the filter and input channel dimensions, and each kernel group $\mathbf{W}_{i,G_{p,q}}$ contains $g_M \times g_N$ kernels, for $p \in P$ and $q \in Q$, where $P = M/g_M$ and $Q = N/g_N$. Therefore, the

regularization term is re-written as

$$R(\alpha_i^{(t)}, \mathbf{W}_i) = \sum_{p=1}^P \sum_{q=1}^Q \sum_{d=1}^{K_d} \sum_{h=1}^{K_h} \sum_{w=1}^{K_w} \|\alpha_{i,G_{p,q}}^{(t)} \odot [\mathbf{W}_{i,G_{p,q}}]_{d,h,w}\|_F^2, \quad (3)$$

where $\alpha_{i,G_{p,q}}^{(t)}$ is updated by $\alpha_{i,G_{p,q}}^{(t)} = \frac{1}{\|[\mathbf{W}_{i,G_{p,q}}]_{d,h,w}^t\|_F^2 + \epsilon}$.

The compression process starts with a pre-trained DNN model. Followed by another training process based on the reweighted regularization pruning algorithm, the pruned model with our block-based pruning constraints can be obtained.

3.3 Compiler: Optimizations

Inspired by PatDNN [16], our compiler-based acceleration relies on several advanced compiler-assisted optimizations to extract the fine-grained structure information in block-based pruned models and generate optimized code that can run on mobile CPU and GPU efficiently.

First, the model weights is stored compactly by leveraging the pruning information (the block pattern), in which further the index arrays can be compressed with a hierarchical structure comparing to the well-known Compressed Sparse Row format. Next, computation reorder is needed to reduce the branches within each thread and eliminate the load imbalance among threads. Blocks are reordered to improve memory and computation regularity, and to eliminate unnecessary memory access. After the reorder, the continuous rows with identical or similar numbers of non-zero weights are processed by multi-threads simultaneously, thus eliminating thread divergence and achieving load balance. Each thread also processes more than one row, thus eliminating branches and improving instruction-level parallelism.

DNN model executions usually involve performance-critical tuning parameters, such as the data placement on GPU memory, matrix tiling sizes, loop unrolling factors, etc. The best-suited configuration of these parameters is hard to be determined manually. Thus, our

compiler also incorporates auto-tuning approaches like TVM [6]. To facilitate an efficient auto-tuning process, we utilize the Genetic Algorithm to explore the best configuration automatically, which allows starting parameter search with initializing an arbitrary number of chromosomes, better exploring the parallelism. Through this auto-tuning process, we can quickly generate acceleration codes for different DNN models and different mobile CPUs/GPUs, which lays the foundation of fast end-to-end latency evaluation.

4 EVALUATION

4.1 Experimental Setup

In our work, our DNN models are trained on a server with eight NVIDIA RTX 2080Ti GPUs. The training methods are implemented using PyTorch API. We evaluate our framework on an off-the-shelf Samsung Galaxy S20 smartphone, containing a Qualcomm Snapdragon 865 Octa-core mobile CPU and a Qualcomm Adreno 650 mobile GPU. The detailed setup is introduced given each specific deep learning tasks.

4.2 Object Detection

State-of-the-art detector YOLO is one of the representative works on object detection. It adopts a unified architecture that extracts feature maps from input images, then it regards the whole feature maps as candidate regions to predict bounding boxes and categories. In our work, YOLOv4 [3] with 320×320 input size is implemented and demonstrated. We denote mAP as the Average Precision under IoU 0.5 threshold and the Frames Per Second (FPS) is measured as the evaluation for the inference speed. The model is trained on MS COCO dataset [10].

The full YOLOv4 model contains 64.36M weight parameters and requires 35.8G floating point operations (FLOPs). Under our block-based pruning, the compression rate can achieve up to 14× (in weight parameters) with 49 mAP, shown in Table 1. Compared with the full model, the weight parameters are reduced to 4.59M and FLOPs are decreased to 3.59G. With 92.87% weight parameters and 88.97% FLOPs pruned, our model still preserves a reasonable accuracy, with only 8.3 mAP loss.

The comparison of different pruning scheme results under 8.1× compression rate is shown in Table 2. Unstructured pruning shows the highest mAP due to its high flexibility. However, the underutilized hardware parallelism limits the inference speed on mobile devices, with only 6.4 FPS. Structured pruning (filter pruning) exhibits a high inference speed, and severe accuracy drop on the downside. Our proposed block-based pruning has achieved both high accuracy and fast inference speed.

Figure 4 shows the end-to-end inference latency comparison of our framework with representative mobile DNN inference acceleration frameworks, including TVM [6], TFLite [21], and MNN [15]. Since these representative frameworks do not support the our block-based sparse model, we also compare the latency of our framework with them using dense (unpruned) YOLOv4 model. As the figure shows, even without incorporating pruning, our framework still outperforms the representative frameworks thanks to our compiler optimization techniques. Under our 14× pruning, our framework achieves 5× speedup compared to the currently best framework MNN on mobile GPU.

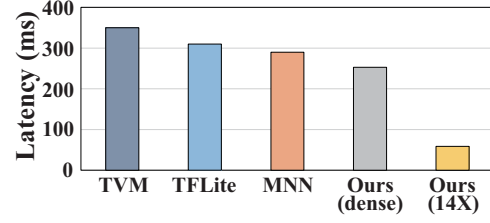


Figure 4: End-to-end object detection inference latency comparisons with representative DNN inference frameworks using mobile GPU.

Table 1: Accuracy and speed under different compression rates on COCO 2014 dataset.

#Weight	#Weights Comp. Rate	#FLOPs	mAP	FPS
64.36M	1.0×	35.8G	57.3	3.5
16.11M	4.0×	10.48G	55.1	7.3
8.04M	8.1×	6.33G	51.4	11.5
6.37M	10.1×	5.48G	50.9	13
4.59M	14.0×	3.95G	49	17

Table 2: Comparison of different pruning schemes (objective detection) .

Pruning Scheme	#Weight	#Weights Comp. Rate	mAP	FPS
Unpruned	64.36M	1×	57.3	3.5
Unstructured	8.04M	8.09×	53.9	6.4
Structured	8.04M	8.09×	38.6	12.7
Ours	8.04M	8.09×	51.4	11.5

4.3 Action Recognition

Our pruning scheme for 3D kernel groups is evaluated three representative 3D CNN models, namely C3D [23], R(2+1)D [22], and S3D [26]. All models are pretrained on the Kinetics dataset [5] and transferred onto UCF101 [20] dataset as the unpruned dense models. The hyperparameter settings are identical for all sparsity schemes. The batch size is 32, and the video clip length is 16 frames. The learning rate is set to $2e-4$ during the pruning process. The total number of epochs is fixed to 240 epochs.

The pruning results of C3D, R(2+1)D, S3D models on UCF101 dataset are provided in Table 3, where our pruning scheme is compared with structured pruning (filter pruning). The latency is measured with the whole video clip containing 16 frames. Combining reweighted regularization with our sparsity scheme, C3D, R(2+1)D, and S3D could achieve only 1%~1.5% accuracy loss on UCF101 under a pruning rate of 2.4~2.6×

As shown in Figure 5, we compared our framework with representative mobile DNN inference acceleration frameworks. For now, only PyTorch Mobile [17] and MNN [15] provide support for the 3D CNNs, and only mobile CPU is supported, where our framework can support both mobile CPU and mobile GPU. The figure shows that our results on both mobile CPU and mobile GPU using the

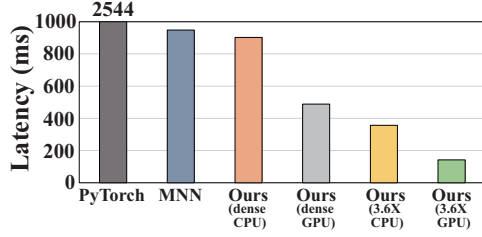


Figure 5: End-to-end action recognition inference latency comparisons with representative DNN inference frameworks using mobile CPU and mobile GPU.

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

Model (Size)	Pruning Scheme	#FLOPs	#FLOPs Comp. Rate	Top-1 Acc. (%)	Latency (ms)
C3D (299MB)	Unpruned	77.2G	1×	81.6	488
	Structured	30.4G	2.6×	79.3	178
	Ours	30.4G	2.6×	80.5	192
	Ours	22.0G	3.6×	80.2	142
R(2+1)D (120MB)	Unpruned	81.5G	1×	94.0	513
	Structured	31.8G	2.6×	90.5	180
	Ours	31.8G	2.6×	92.5	201
	Ours	26.2G	3.2×	92.0	141
S3D (31MB)	Unpruned	49.4G	1×	90.6	565
	Structured	24.8G	2.1×	88.2	325
	Ours	24.8G	2.1×	90.2	342
	Ours	20.6G	2.4×	89.1	293

both dense model and pruned model consistently outperform other frameworks. The acceleration is up to 6.7× when compared to the currently best framework MNN.

4.4 Medical Diagnosis

DNNs have also made great help in the field of medical diagnosis, such as pneumonia diagnosis and localization. We developed a DNN model trained on a dataset provided by RSNA Pneumonia Detection Challenge [1]. In the pre-processing stage, the labeled images are resized to 320×320 , along with the corresponding coordinates of bounding boxes. YOLOv3 [19] is adopted as the base feature detector with our customized anchor box priors, due to the speed boost and high average precision.

The original model has 61.47M weight parameters and requires 38.63G FLOPs. Under our framework, the compression rate can achieve 11.6×, with only 1.0 mAP loss, achieving 15.5 FPS. Moreover, we compare unstructured pruning, structured pruning (column pruning) and our block-based pruning under the same compression rate. The results are shown in Table 4, which indicates the outperformance of our pruning scheme.

5 CONCLUSION

In this work, our unified mobile acceleration of DNNs framework is introduced, achieving real-time computer vision on off-the-shelf mobile devices. The framework starts with our fine-grained block-based weight pruning, and followed by our unique compiler optimizations accommodating the unique structured pruning properties. Three representative fields of applications are implemented

Table 4: Comparison of different pruning schemes (medical diagnosis).

Pruning Schemes	#Weights Comp. Rate	#FLOPs	mAP	FPS
Unpruned	1×	38.63G	71.2	4.2
Unstructured	11.6×	3.63G	70.5	10.8
Structured	11.6×	3.63G	69.7	16.7
Ours	11.6×	3.63G	70.2	15.5

and demonstrated, approving the effectiveness of our proposed framework.

ACKNOWLEDGMENTS

This work is funded by the National Science Foundation Awards CCF-1937500 and CNS-1909172.

REFERENCES

- [1] 2018. *RSNA Pneumonia Detection Challenge*. <https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/overview>
- [2] Michael G Bechtel, Elise McElhiney, Minje Kim, and Heechul Yun. 2018. Deepcar: A low-cost deep neural network-based autonomous car. In *RTCSA*.
- [3] Alexey Bochkovskiy et al. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv preprint arXiv:2004.10934* (2020).
- [4] Emmanuel J Candes et al. 2008. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier analysis and applications* (2008).
- [5] Joao Carreira and Andrew Zisserman. 2017. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*.
- [6] Tianqi Chen et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *OSDI*.
- [7] Sharan Chetlur et al. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).
- [8] Yiwen Guo et al. 2016. Dynamic network surgery for efficient dnns. In *NeurIPS*.
- [9] Yihui He et al. 2017. Channel pruning for accelerating very deep neural networks. In *ICCV*.
- [10] Tsung-Yi Lin et al. 2014. Microsoft coco: Common objects in context. In *ECCV*.
- [11] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the Value of Network Pruning. In *ICLR*.
- [12] Xiaolong Ma et al. 2019. Non-Structured DNN Weight Pruning – Is It Beneficial in Any Platform?
- [13] Xiaolong Ma et al. 2020. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In *AAAI*.
- [14] Chuhan Min et al. 2018. 2pfpc: Two-phase filter pruning based on conditional entropy. *arXiv preprint arXiv:1809.02220* (2018).
- [15] Alibaba MNN. [n.d.]. <https://github.com/alibaba/MNN>.
- [16] Wei Niu et al. 2020. Patdnn: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning. In *ASPLOS*.
- [17] PyTorch. 2019. <https://pytorch.org/mobile/home>.
- [18] Zhaofan Qiu, Ting Yao, and Tao Mei. 2017. Learning spatio-temporal representation with pseudo-3d residual networks. In *ICCV*.
- [19] Joseph Redmon and Ali Farhadi. 2018. Yolo3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).
- [20] Khuram Soomro et al. 2012. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402* (2012).
- [21] TensorFlow-Lite. [n.d.]. https://www.tensorflow.org/lite/performance/model_optimization.
- [22] Du Tran et al. 2018. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*.
- [23] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. 2015. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*.
- [24] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *NeurIPS*.
- [25] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *ECCV*.
- [26] Saining Xie et al. 2018. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *ECCV*.
- [27] Chaoyun Zhang et al. 2019. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials* (2019).
- [28] Tianyun Zhang et al. 2018. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *ECCV*.