Towards Real-Time DNN Inference on Mobile Platforms with Model Pruning and Compiler Optimization

Wei Niu1*, Pu Zhao2*, Zheng Zhan2*, Xue Lin2, Yanzhi Wang2 and Bin Ren1

1College of William and Mary
2Northeastern University
wniu@email.wm.edu, {zhao.pu, zhan.zhe}@husky.neu.edu, {xue.lin, yanz.wang}@northeastern.edu, bren@cs.wm.edu

Abstract

High-end mobile platforms rapidly serve as primary computing devices for a wide range of Deep Neural Network (DNN) applications. However, the constrained computation and storage resources on these devices still pose significant challenges for real-time DNN inference executions. To address this problem, we propose a set of hardware-friendly structured model pruning and compiler optimization techniques to accelerate DNN executions on mobile devices. This demo shows that these optimizations can enable real-time mobile execution of multiple DNN applications, including style transfer, DNN coloring and super resolution.

1 Introduction

There are two key phenomena in machine learning and mobile computing fields. First, various Deep Neural Networks (DNN) have served as the fundamental building block of a broad spectrum of machine learning applications because of its superior accuracy and self adaptiveness ability [Goodfellow et al., 2016]. Second, with the rapidly increasing popularity of mobile phones, high-end mobile platforms (rather than desktops or servers) serve as primary computing devices, especially for many DNN applications such as wearable devices, video streaming, smart health devices, etc. [Philipp et al., 2011; Lane et al., 2015].

It is desirable to deploy real-time DNN inference systems on mobile platforms. However, due to the intensive computation and high memory storage requirements of state-of-the-art DNN models, such as VGG-16 [Simonyan and Zisserman, 2014], ResNet-50 [He et al., 2016] and MobileNet [Howard et al., 2017], it is quite challenging to achieve real-time DNN executions on mobile devices.

Multiple end-to-end mobile DNN acceleration frameworks have been developed, such as TVM [Chen et al., 2018], TensorFlow-Lite (TFLite) [Tensorflow Lite, 2017], and Alibaba Mobile Neural Network (MNN) [Mobile Neural Network, 2019]. However, they still cannot satisfy the real-time execution requirement on mobile devices. TVM takes 198 ms to complete the inference of a video frame on an embedded GPU (Adreno 640) with VGG-16, which is important in transfer learning. It takes even longer by TFLite (268 ms).

This paper investigates DNN inference on mobile platforms with an ultimate goal of real-time execution. The contributions of this paper are summarized as follows:

- First, it proposes a set of hardware-friendly structured model pruning and compiler optimization techniques to accelerate DNN executions on mobile devices.
- Second, it implements and accelerates three interesting and key DNN applications, style transfer [Gatys et al., 2016], DNN coloring [Iizuka et al., 2016], and super resolution [Dong et al., 2014] with the help of the proposed model pruning and compiler optimizations.
- Third, it demonstrates that these three applications can achieve real-time executions on mobile devices. To the best of our knowledge, our implementations are the fastest on mobile devices.

2 Structured Model Pruning

To satisfy the constraints of computation and storage on mobile, various DNN model compression methods are proposed, where weight pruning [Luo and Wu, 2017; Mao et al., 2017; Wen et al., 2016] leads to a notable model size reduction.

The non-structured weight pruning [Luo and Wu, 2017; Guo et al., 2016] is not friendly to modern hardware designs as the indices of the sparse model weights result in stall or complex workload on parallel (specifically, massive parallel) architectures [Wen et al., 2016]. We mainly explore the more hardware-friendly structured weight pruning [Wen et al., 2016], which stores the pruned model regularly in its shape without any weight indices.

Multiple structured pruning approaches exist based on their pruning dimensions, including filter pruning (that prunes the whole filter), channel pruning (that prunes channels), column pruning (that prunes the same location in each filter of each layer), and connectivity and pattern pruning (that prunes both the channels and certain locations in each kernel simultaneously) [Liu et al., 2019; Ma et al., 2019]. Despite the differences of these structured pruning methods, we support them on a uniform framework based on Alternating Direction Method of Multipliers (ADMM) [Boyd et al., 2011; Zhao et al., 2019b; Weng et al., 2020]. In general, the pruning

---

*These authors contributed equally
optimization problem is formulated as,

$$\min_{\{W_i\}} f(\{W_i\}), \text{subject to } W_i \in S_i, \forall i$$ (1)

where $f$ denotes the loss function, $\{W_i\}$ represents the model weights, and $S_i$ shows the constraint of the remaining weights in the $i$-th layer to satisfy a certain structure (e.g., pre-defined patterns or certain columns/rows preservation). Although the structure constraints make the problem non-differentiable and more complicated, ADMM is able to split the original problem into several easier sub-problems, and iteratively solve them until convergence [Zhao et al., 2018; Zhao et al., 2019a]. We apply column pruning for style transfer and kernel pruning for coloring and super resolution.

3 Compiler Optimization

Compiler optimizations consist of three components:

**DSL related optimization.** A DNN model comprises multiple operators (layers) that may show varied computation patterns. A new DSL (i.e., domain specific language) is designed to represent DNN models. This DSL employs a new LR (i.e., layer-wised representation) to represent each layer. Essentially, this DSL is equivalent to the computational graph. Some further computational graph transformation optimizations are also applied to this DSL (e.g., a combination of Convolution layer/Depthwise Convolution layer + BatchNorm layer + Activation layer) to reduce the data movement and increase instruction level parallelism.

**Sparse model storage.** To further improve data locality, the weights of the sparse model are also stored in a more compact format than well-known CSR. This sparse model storage aims to avoid zero-weights storage as CSR with an even better compression ratio by further removing redundant indices generated by our structured pruning. It helps to save the scarce memory-bandwidth of mobile devices.

**Matrix reorder.** Structured pruning eventually transforms model kernel matrices into small blocks with different pruning patterns. Without any further optimizations, well-know challenges for sparse matrix multiplications still exist, i.e., heavy load imbalance among each thread, and irregular memory accesses. To address this issue, a matrix reorder approach is proposed by leveraging the structure information offered by our pruning. For example, if a column (and row) pruning is applied. Because this pruning removes all kernel weights in certain columns and rows within a block, the remaining weights only appear in other rows and columns with a certain degree of regularity. Based on this insight, matrix reorder first reorders the rows (e.g., filters in CNN) by arranging the ones with the same or similar patterns together. Next, it compacts the weights in the column direction (e.g., kernels in CNN).

4 Experiments and Demonstrations

This experiment demonstrates the efficacy of the proposed pruning and compilation acceleration approach through three interesting and important DNN applications, style transfer [Gatys et al., 2016], DNN coloring [Iizuka et al., 2016], and super resolution [Dong et al., 2014]. The style transfer model is based on a generative network [Zhang and Dana, 2017] trained on Microsoft COCO [Lin et al., 2014]. DNN coloring uses the Places scene [Zhou et al., 2014] dataset to train a novel architecture that can jointly extract and fuse global and local features to perform the final colorization. The super resolution model mainly utilizes residual blocks with wider activation and linear low-rank convolution [Yu et al., 2018] trained on the DIV2K [Timofte et al., 2017] dataset. With structured pruning and compiler optimization, we implement the models on a Samsung Galaxy S10 mobile phone. We demonstrate that our implementations are able to achieve real-time inference on mobile with video demos.

Figure 1 shows sample input and output of three applications. Table ?? shows the average inference time of the applications on mobile device. Structured pruning and compiler optimization accelerate the inference with speedups of 4.2×, 3.6×, and 3.7× for style transfer, coloring and super resolution, respectively. These results demonstrate that our optimized implementation generates satisfied output with high speed. More specifically, all inference can complete within 75 ms, showing the possibility of achieving real-time executions of complex DNN applications on mobile. Please find more video demos at our YouTube or bilibili channel12.

5 Potential Impacts

Real-time DNN executions on mobile have great potential to impact many fields. Take real-time super resolution as an example. It enables users to enjoy high-resolution video streams with limited network bandwidth and inexpensive data cost, while saving providers storage and communication resources.

---

1. [www.youtube.com/channel/UCCKVDTg2eheRTEuqJ5cD8A/](www.youtube.com/channel/UCCKVDTg2eheRTEuqJ5cD8A/)
2. [space.bilibili.com/573588276?from=search&seid=13333469485394270447](space.bilibili.com/573588276?from=search&seid=13333469485394270447)
Because only low resolution video streaming is required to store and communicate, super resolution results in significant video service improvement and commercial cost reduction.

Acknowledgments

This work is partly supported by the National Science Foundation grants CNS-1932351, CCF-1919117, and CNS-1909172.

References


