ADMM-based Weight Pruning for Real-Time Deep Learning Acceleration on Mobile Devices

Hongjia Li
li.hongjia@husky.neu.edu
Northeastern University
Boston, USA

Ning Liu
Xiaolong Ma
Sheng Lin
Northeastern University
Boston, USA

Shaokai Ye
Tianyun Zhang
Syracuse University
Syracuse, USA

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

Wenyao Xu
wenyaoxu@buffalo.edu
University at Buffalo
Buffalo, USA

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

ABSTRACT

Deep learning solutions are being increasingly deployed in mobile applications, at least for the inference phase. Due to the large model size and computational requirements, model compression for deep neural networks (DNNs) becomes necessary, especially considering the real-time requirement in embedded systems. In this paper, we extend the prior work on systematic DNN weight pruning using ADMM (Alternating Direction Method of Multipliers). We integrate ADMM regularization with masked mapping/retraining, thereby guaranteeing solution feasibility and providing high solution quality. Besides superior performance on representative DNN benchmarks (e.g., AlexNet, ResNet), we focus on two new applications facial emotion detection and eye tracking, and develop a top-down framework of DNN training, model compression, and acceleration in mobile devices. Experimental results show that with negligible accuracy degradation, the proposed method can achieve significant storage/memory reduction and speedup in mobile devices.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; **Real-time simulation**;

KEYWORDS

Mobile devices, neural networks, acceleration, real-time

ACM Reference Format:
Hongjia Li, Ning Liu, Xiaolong Ma, Sheng Lin, Shaokai Ye, Tianyun Zhang, Xue Lin, Wenyao Xu, and Yanzhi Wang. 2019. ADMM-based Weight Pruning for Real-Time Deep Learning Acceleration on Mobile Devices. In *Great Lakes Symposium on VLSI 2019 (GLSVLSI '19), May 9–11, 2019, Tysons Corner, VA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3299874.3319492

GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA

ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145/3299874.3319492

1 INTRODUCTION

Recently, deep learning has been expanded into many new application fields, such as automatic drive system, 3D printing detection, and medical imaging and diagnosis [15, 19, 25]. As an example for the latter application, deep neural networks (DNNs) can be trained for the detection of facial expressions and performing eye tracking for the patients [6, 16]. By extracting complex and high-level features from large-scale data, DNNs can achieve a high accuracy and provide significant help and convenience for both doctors and patients.

DNNs are typically trained in an offline manner, and are often deployed in low-power, embedded, or mobile devices during inference. One of the major challenges is the large model size and computational requirement, which makes it difficult for real-time implementation in mobile devices. To overcome this challenge, many efforts have been devoted to DNN model compression from both industry and academia. One pioneering work [9] adopts an iterative heuristic for DNN weight pruning, achieving good pruning results: 9× weight reduction in AlexNet [17] and 12× in LeNet-5 [18]. Despite the promising results, the compression gain mainly focuses on the fully-connected (FC) layers, and the pruning ratio is limited on convolutional (CONV) layers (e.g., 2.7× for CONV layers in AlexNet). This limitation needs to be overcome as CONV layers become the most computationally intensive layers in current DNNs [11, 17].

Later weight pruning work extend to (i) use more sophisticated heuristic such as both weight prune and grow [4, 8], (ii) strike a desirable tradeoff between pruning ratio and accuracy, and (iii) incorporate regularity or structure in weight pruning framework [12, 26]. To partially overcome the heuristic nature, recently, a systematic DNN weight pruning framework has been proposed in [27], based on the powerful ADMM (Alternating Direction Methods of Multipliers) technique [1]. This work formulates the DNN weight pruning problem as a mathematical optimization problem, and observes the compatibility between the combinatorial constraints (associated with weight pruning) with ADMM. It achieves improved weight pruning results, 21× on AlexNet and 71.2× on LeNet-5, with no accuracy loss. However, this work adopts a direct application of ADMM, and lacks rigorous guarantee on feasibility (satisfying all constraints) and solution quality due to the non-convex objective function.

Using [27] as the starting point, this work first develops a systematic, algorithmic DNN weight pruning framework, with an integration of ADMM regularization and masked mapping/retraining steps. In this way the solution feasibility can be guaranteed and solution quality (test accuracy) can be improved. The proposed framework outperforms prior work on representative benchmarks such as AlexNet and LeNet-5, and also achieves 18× weight reduction on ResNet-50 [11], which is widely accepted to be difficult for compression. We further incorporate structured pruning [26], including filter-wise, channel-wise, and filter shape-wise sparsities, into the ADMM regularization framework, thereby facilitating high parallelism and hardware implementations.

Deep learning solutions are being increasingly deployed in mobile applications, at least for the inference phase[20]. Due to the large model size and computational requirements, model compression for deep neural networks (DNNs) becomes necessary, especially considering the real-time requirement in embedded systems. For real-world and real-time applications, we apply the proposed framework on two medical-related applications and implement on different embedded systems. The first application is the facial emotion recognition [6]. The second application is eye tracking [16]. DNNs for both applications are mainly CONV layers. The first application uses the FER-2013 dataset [6], and we prune 78.33% of total weights with (almost) no accuracy loss. We can achieve 97.2% weight reduction in the second application. For inference acceleration in mobile devices, we use sparse matrices and *dictionary of keys*[5] for weight representation and computation after pruning, and perform testing on three mobile devices. We achieve 10x speedup and 13.2x speedup on these two applications, respectively, when testing on mobile devices. Our models are released at the anonymous link https://bit.ly/2P1ehdb.

2 A SYSTEMATIC WEIGHT PRUNING FRAMEWORK USING ADMM

2.1 Systematic View of Weight Pruning

Similar to [27], we provide a systematic view of DNN weight pruning during training as an optimization problem. Consider a general N-layer DNN. Sets of weights and biases of the i-th (CONV or FC) layer are denoted by $\mathbf{W}_i$ and $\mathbf{b}_i$, respectively. Let us denote the loss function of the N-layer DNN as $f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big)$. Then the overall problem is defined as

$$\begin{aligned}&\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}}\, f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big),\\ &\text{subject to } \mathbf{W}_i \in \mathcal{S}_i,\ i = 1, \ldots, N.\end{aligned} \tag{1}$$

The set $\mathcal{S}_i = \big\{\mathbf{W}_i \big| \text{card}(\text{supp}(\mathbf{W}_i)) \le \alpha_i\big\}$ reflects constraint for weight pruning, where 'card' refers to cardinality and 'supp' refers to support set. Elements in $\mathcal{S}_i$ are $\mathbf{W}_i$ solutions, satisfying that the number of non-zero elements in $\mathbf{W}_i$ is limited by α_i for layer i. Because of such combinatorial constraints, problem (1) cannot be solved using conventional stochastic gradient descent which assumes no hard constraints. This is key reason that prior work use heuristic methods to get rid of these constraints. To overcome this limitation, a key observation is that such form of combinatorial constraints is compatible with ADMM technique.

2.2 Connection to ADMM

ADMM [23, 24] is a powerful optimization tool, by decomposing an original problem into two subproblems that can be solved separately and iteratively. Consider optimization problem

$$\min_{\mathbf{x}} \ f(\mathbf{x}) + g(\mathbf{x}). \tag{2}$$

In ADMM, the problem is first re-written as

$$\min_{\mathbf{x},\mathbf{z}} \ f(\mathbf{x}) + g(\mathbf{z}), \ \ \text{subject to} \ \ \mathbf{x} = \mathbf{z}. \tag{3}$$

Next, by using augmented Lagrangian [1], the above problem is decomposed into two subproblems on $\mathbf{x}$ and $\mathbf{z}$. The first is $\min_{\mathbf{x}} f(\mathbf{x}) + q_1(\mathbf{x}|\mathbf{z})$, where $q_1(\mathbf{x}|\mathbf{z})$ is a quadratic function of $\mathbf{x}$ with fixed $\mathbf{z}$. Subproblem 2 is $\min_{\mathbf{z}} g(\mathbf{z}) + q_2(\mathbf{z}|\mathbf{x})$, where $q_2(\mathbf{z}|\mathbf{x})$ is a quadratic function on $\mathbf{z}$ with fixed $\mathbf{x}$. The two subproblems will be solved iteratively until convergence is achieved [23, 24].

ADMM is conventionally utilized to accelerate convergence of convex optimization problems. The optimality and fast convergence have been proven for convex problems [1, 23]. As a special property, ADMM can effectively deal with a subset of combinatorial constraints and yields optimal (or at least high quality) solutions [14, 21]. Our observation is that associated constraints in DNN weight pruning belong to this subset. More specifically, we use indicator functions to incorporate combinatorial constraints into objective function. The indicator functions are $g_i(\mathbf{W}_i) = \begin{cases} 0 & \text{if } \mathbf{W}_i \in \mathcal{S}_i \\ +\infty & \text{otherwise} \end{cases}$. Then problem (1) becomes:

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \ f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big) + \sum_{i=1}^N g_i(\mathbf{W}_i) \tag{4}$$

Despite the compatibility of the combinatorial constraints with ADMM, there is difficulty in using ADMM directly due to the non-convex nature of the objective function in (1). Therefore, special mechanisms are needed to guarantee the *solution feasibility* and *solution quality*.

2.3 Systematic DNN Weight Pruning

Instead of direct application of ADMM, we develop an integrated framework of *ADMM regularization* and *masked mapping and retraining* as showed in Algorithm 1. We guarantee solution feasibility (satisfying all constraints) and provide high quality (maintaining test accuracy).

The ADMM regularization starts from a DNN model without compression. By incorporating auxiliary variables $\mathbf{Z}_i$'s, and dual variables $\mathbf{U}_i$'s, we decompose (4) into two subproblems, and iteratively solves them until convergence. The first subproblem is

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \ f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big) + \sum_{i=1}^N \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i^k + \mathbf{U}_i^k\|_F^2. \tag{5}$$

The first term in (5) is the differentiable (non-convex) loss function of the DNN, while the other quadratic terms are differentiable and convex. As a result, this subproblem can be solved by stochastic gradient descent similar to the one that would be used to train the original DNN.

The second subproblem is

$$\underset{\{\mathbf{Z}_i\}}{\text{minimize}} \quad \sum_{i=1}^{N} g_i(\mathbf{Z}_i) + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i^{k+1} - \mathbf{Z}_i + \mathbf{U}_i^k\|_F^2. \tag{6}$$

The optimal, analytical solution is the Euclidean projection of $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ onto the set $\mathcal{S}_i$. Since α_i is the desired number of weights after pruning in the i-th layer, we can prove that the Euclidean projection results in keeping α_i elements in $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ with the largest magnitudes and setting the remaining weights to zeros. After both subproblems solved, we update the dual variables $\mathbf{U}_i$ as Eqn. (7) and complete one iteration in ADMM regularization.

$$\mathbf{U}_i^{k+1} = \mathbf{U}_i^k + \mathbf{W}_i^{k+1} - \mathbf{Z}_i^{k+1} \tag{7}$$

Masked Mapping and Retraining: We extend the formulation in [27] by introducing masked mapping and retraining step. After ADMM regularization, we obtain intermediate $\mathbf{W}_i$ solutions. In this step, we first perform the said Euclidean projection (mapping) to guarantee that at most α_i weights in each layer are non-zero. Next, we mask the zero weights and retrain the DNN with non-zero weights using training sets (while keeping the masked weights 0). In this way test accuracy can be partially restored.

Algorithm 1 shows the pseudo-codes of the proposed weight pruning training algorithm, in which *ADMM_ITERATION* is normally set as $\frac{1}{10}$ of *MAX_ITERATION_FOR_SGD*.

Algorithm 1: Systematic Weight Pruning Training Algorithm using ADMM

```
Initialize training hyperparameters;
for CURRENT_ITERATION <
  MAX_ITERATION_FOR_SGD do
    Solve(Eqn. (5));
    if CURRENT_ITERATION % ADMM_ITERATION == 0
     then
        Solve(Eqn. (6));
        Solve dual update according to Eqn. (7);
    end
end
Masked mapping;
Retrain the pruned model;
```

Feasibility and Solution Quality: One can observe that constraints on weight pruning are satisfied through the mapping step and that the retraining process restores the accuracy loss of mapping. ADMM regularization acts as a smart, adaptive DNN regularization (see Eqn. (5)), where the regularization targets are dynamically updated in each iteration by solving subproblem 2 (optimally and analytically). This is one key reason that this method outperforms many prior works on DNN weight pruning based on fixed regularization [26], where regularization targets are not updated.

Sample Results on Representative DNNs: Sparse matrices are employed for representing the pruned weights due to the reduced space complexity (by storing only the non-zero entries and index to the next non-zero entry) and associated computation savings. We have performed testing on representative DNNs, LeNet-5 [18] for MNIST dataset, and AlexNet [17] and ResNet-50 [11] for ImageNet dataset. As shown in Table 1, we achieve 167x reduction in number of weights in LeNet-5, 31x in AlexNet, and 15x in ResNet-50, with (almost) no accuracy loss. These results consistently outperform prior arts especially on ResNet, which is difficult for pruning in prior work.

Table 1: Sample results on representative DNNs

Network	Method	Accuracy Loss	Weights	Pruning Rate
LeNet-5 (99.2%)	Network Pruning [10]	0.0%	34.5K	12.5x
	Direct ADMM[27]	0.0%	6.05K	71.2x
	Our Proposed Method	0.2%	2.58K	**167**x
AlexNet (80.2%)	Network Pruning [10]	0.0%	6.7M	9x
	NEST [4]	0.0%	3.9M	15.7x
	Direct ADMM[27]	0.0%	2.9M	21x
	Our Proposed Method	0.1%	1.9M	**31**x
ResNet-50 (92.4%)	Fine-grained Pruning [22]	0.1%	12.6M	2.6x
	Our Proposed Method	0.4%	2.18M	**15**x
		0.8%	1.82M	**18**x

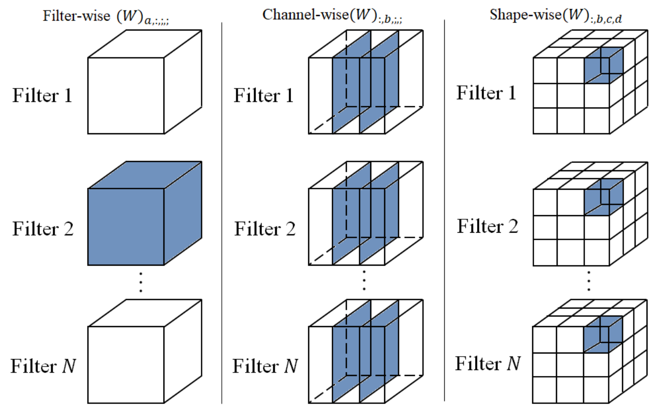

Figure 1: Examples of filter-wise, channel-wise and shape-wise structured sparsities.

2.4 Incorporating Structures in Weight Pruning

As discussed before, the DNN after weight pruning is an irregular, sparse neural network, and sparse matrices with indices are utilized for weight storage. One clear disadvantage is the limitation on parallelism degree and therefore degradation in hardware performance as also observed in [26]. Prior work (e.g., [26]) incorporates regularity or "structures" in DNN weight pruning in order to solve this problem, but lacks a systematic approach to achieve this goal.

We make an observation that structured pruning is compatible with the ADMM-based weight pruning framework, and therefore can be solved systematically. We use CONV layers (the most computationally intensive in current DNNs [11, 17]) as an illustrative example while FC layers can be treated similarly. There are three types of structured sparsities, *filter-wise*, *channel-wise*, and *shape-wise* sparsities as shown in Figure 1.

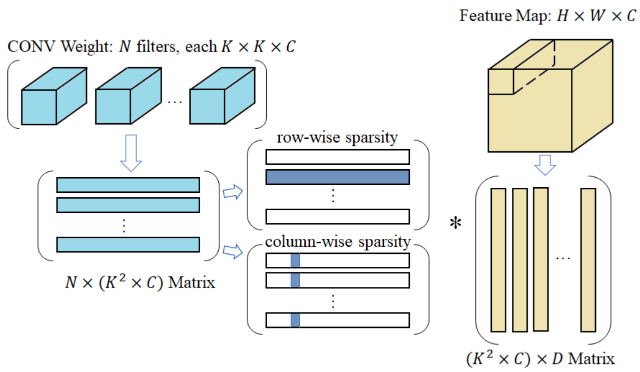

Figure 2: Examples of GEMM in CONV layer and effect of structured sparsities.

CONV operations in DNNs are commonly transformed to matrix multiplications by converting weight tensors and feature map tensors to matrices [3], named *general matrix multiplication* or GEMM, as shown in Figure 2, in order to facilitate implementation from mobile devices to GPUs. Filter-wise pruning corresponds to reducing number of rows, while channel-wise and filter shape-wise prunings correspond to reducing the number of columns. As a result, a combination of the said three structured sparsities will reduce the dimension in GEMM while maintaining a full matrix, thereby facilitating acceleration in mobile/hardware platforms.

All the above three structured pruning scenarios can be incorporated into the ADMM regularization framework. For filter-wise sparsity as an example, the constraint set $\mathcal{S}_i$ will indicate that the number of nonzero filters in $\mathbf{W}_i$ is less than a predefined value. For channel-wise sparsity, constraint set $\mathcal{S}_i$ will indicate that the number of nonzero channels in $\mathbf{W}_i$ is less than a predefined value. In this way structured pruning will replace the second subproblem in (6), and the optimal Euclidean mapping will be derived accordingly. The first subproblem in (5) and the masked mapping/retraining step will maintain the same form (and solution method).

3 MOBILE IMPLEMENTATION/ ACCELERATION OF PRUNED DNNS

In this section, we describe mobile implementation/ acceleration of the inference phase of pruned DNNs. As shown in Figure 3, there are four high-level modules on Android-based platforms: model constructor, parameters loader, dataset loader and inference engine. In the first module, the network architecture will be constructed and through parameters import, pre-trained weights and bias will be loaded. In the third module, the test data are loaded through camera or files and we perform inference in the last module through C++ interface.

Since GEMM is utilized for CONV layer computation in mobile systems, and CONV layer is the most computationally intensive layer, we adopt sparse matrices in GEMM based on the weight pruning results. Specifically, the computational complexity is reduced from $O(N^3)$ to $O(K \cdot N)$, where N is weight matrix dimension, and K is the number of non-zero elements in the matrix. We implement sparse matrix computation in a bottom-up manner from C++ array

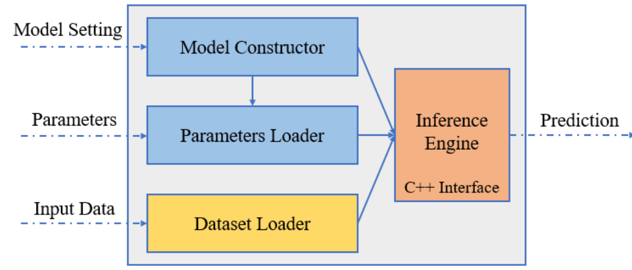

Figure 3: Building blocks of software implementation.

template, instead of existing libraries (e.g., *OpenCV* [2], *Eigen* [7]). This is because of the limited scalability of the mobile version of such libraries to support large-scale DNNs.

The DNN inference acceleration algorithm is shown in Algorithm 2, focusing on GEMM acceleration for a specific layer. In each layer, the input matrix and weight matrix in GEMM are denoted by $\mathbf{X}$ and $\mathbf{W}$, respectively. The bias vector is $\mathbf{b}$. To accelerate the computation, the structure of the weight matrix $\mathbf{W}$ is transformed to the sparse matrix structure denoted as $\mathbf{W}'$, in which *Dictionary of Keys* is adopted to efficiently construct and represent. *Step 5* to *12* show the details of the general multiplication of the dense input matrix $\mathbf{X}$ and the sparse weight matrix $\mathbf{W}'$. The dimensionality of $\mathbf{W}'$ is $K \times 3$.

Algorithm 2: Pruned Model Inference Algorithm on Mobile Devices (focusing on GEMM for one layer)

1 **Input:** $m \times n$ matrix $\mathbf{X}$;
2 **Parameters:** $\mathbf{W}, \mathbf{b}$;
3 **Output:** matrix $\mathbf{Y}$;
`/* convert dense matrix to sparse matrix */`
4 $\mathbf{W}' \leftarrow \mathbf{W}$;
`/* sparse matrix * dense matrix operation */`
5 **for** $index \leftarrow 0$ *until* K **do**
6   $row \leftarrow \mathbf{W}'_{index,0}$;
7   $col \leftarrow \mathbf{W}'_{index,1}$;
8   $value \leftarrow \mathbf{W}'_{index,2}$;
9   **for** $j \leftarrow 0$ *until* n **do**
10    $\mathbf{Y}_{row,j} \leftarrow \mathbf{Y}_{row,j} + value * \mathbf{X}_{col,j}$;
11   **end**
12 **end**
13 $\mathbf{Y} \leftarrow \mathbf{Y} + \mathbf{b}$;
14 Return $\mathbf{Y}$;

4 EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, we perform evaluation on two medical related applications, starting from DNN construction, systematic weight pruning using ADMM, and finally implementation of DNN inference on

Table 2: Platforms under test and specifications.

Platform	Android	Primary CPU	Companion CPU	CPU Architecture	GPU	RAM (GB)
Huawei Honor 6X	7 (Nougat)	4 × 2.1GHz Cortex-A53	4 × 1.7GHZ Cortex-A53	ARMv8-A	Mali T830	3
LG Nexus 5X	8.1 (Oreo)	4 × 1.4 GHz Cortex-A53	2 × 1.8GHz Cortex-A57	ARMv8-A	Adreno 418	2
Huawei Honor 10	8.1 (Oreo)	4 × 2.4 GHz Cortex-A73	4 × 1.8 GHz Cortex-A53	ARMv8-A	Mali-G72 MP12	6

multiple mobile devices to evaluate the applicability of the inference phase. Table 2 summarizes the specifications of test mobile platforms.

The performance evaluation includes two aspects: (a) storage reduction due to the reduced DNN model size, (b) inference acceleration (running time reduction). The largest portion of inference time is from GEMM operation, which will benefit from sparse matrices. We compare the run-time cost of GEMM as well as the overall run-time.

4.1 Case I: Facial Emotion Recognition

Facial emotion recognition is utilized in many fields such as medical, entertainment and security. For this application, FER-2013 face database [6] is used to train and test the DNN model. The dataset comprises a total of 35,887 pre-cropped, 48-by-48-pixel grayscale images of faces each labeled with one of the 7 emotion classes: anger, disgust, fear, happiness, sadness, surprise, and neutral. Due to the small portion of the disgust class, the dataset is merged into six classes including angry, fear, happy, sad, surprise, and neutral [13].

A typical DNN is constructed, comprising 9 CONV layers with one max-pooling after every three CONV layers. There are 32, 64, and 218 filters in these three CONV-layer groups, respectively. In addition, 2 FC layers are constructed followed by a softmax layer in the end. Table 3 shows the weight pruning result. Our pruning algorithm can achieve a high compression ratio by 4.6x without accuracy loss.

Table 3: Weight Pruning Result on Facial Emotion Recognition Model

Layer	Weights	Weights after prune	Matrices sparsity
conv1	288	230	20%
conv2	9216	3687	60%
conv3	9216	2765	70%
conv4	18432	3687	80%
conv5	36864	7373	80%
conv6	36864	5530	85%
conv7	73728	11060	85%
conv8	147456	22119	85%
conv9	147456	22119	85%
fc1	294912	88474	70%
fc2	4096	1639	60%
fc3	384	154	60%
Total	778912	168837	78.33%

In Table 4, the overall performance is shown including the accuracy and inference time on various mobile systems. According to the results, the overall acceleration of the pruned model can achieve up to 10x, while the GEMM acceleration can achieve up to 22x compared with the non-pruned model. The actual speedup is even higher than the pruning ratio, due to two reasons: (i) we focus more on the pruning of computationally intensive CONV layers than FC layers, and (ii) the bandwidth requirement is also reduced besides computation reduction.

Table 4: Performance of Facial Emotion Recognition Model on Mobile Devices

		Model	Pruned model	Speedup
Accuracy		58.2056%	58.0663%	-
Honor 6X	Overall	0.971s	0.133s	7.3x
	GEMM	0.925s	0.082s	11.3x
Nexus 5X	Overall	0.161s	0.015s	10x
	GEMM	0.137s	0.006s	22x
Honor 10	Overall	0.304s	0.046s	6.6x
	GEMM	0.283s	0.025s	11.3x

4.2 Case II: Eye Tracking

Eye tracking is one widely used application in many areas such as human-computer interaction, medical diagnoses, psychological studies and computer vision. To implement the weight pruned eye tracking model for embedded systems, we use GazeCapture, a large-scale mobile eye tracking dataset, containing data from over 1,450 people with almost 2.5M frames [16].

The eye tracking DNN model takes as input the detected and cropped portions of the original frame, including left eye, right eye, and face images (all of size 224 x 224). Additionally, the face grid is considered as another input, with a binary mask to indicate the location and size of the head within the frame (of size 25 x 25). The output is the distance, in centimeters, from the camera. The whole model consists of 13 CONV layers and 7 FC layers. Compared with typical DNN, the eye tracking network has a more complicated architecture. It consists of three small typical DNNs taking inputs from the right eye, left eye and face respectively. The outputs get concatenated and go through 7 FC layers along with the input from face grid. The details of model structure are demonstrated in [16]. The network architecture also shows a big impact on the overall real-time speedup.

After applying our weight pruning method, the weight reduction result is shown in Table 5. The overall pruning ratio can achieve 36x. The total number of weights is reduced from 718K to 19K. The performance on three mobile devices is demonstrated in the following Table 6. The pruned model achieves up to 13x speedup compared with the non-pruned model, especially, the GEMM computing time gets accelerated up to 28x.

Table 5: Weight Pruning Result of Eye Tracking Model

Layer	Weights	Weights after prune	Matrices sparsity
conv-e1	4875	3413	30%
conv-e2	102400	512	99.5%
conv-e3	73728	369	99.5%
conv-e4	8192	1229	85%
conv-f1	4875	3413	30%
conv-f2	102400	512	99.5%
conv-f3	73728	369	99.5%
conv-f4	8192	1229	85%
fc-e1	65536	3933	94%
fc-f1	32768	656	98%
fc-f2	8192	1475	82%
fc-fg1	160000	800	99.5%
fc-fg2	32768	984	97%
fc1	40960	615	98.5%
fc2	256	256	-
Total	718870	19765	97.25%

Table 6: Performance of Eye Tracking Model on Mobile Devices

		Model	Pruned model	Speedup
Accuracy		92.97%	91.47%	-
Honor 6X	Overall	0.563s	0.043s	13.2x
	GEMM	0.541s	0.019s	28.8x
Nexus 5X	Overall	0.070s	0.009s	7.5x
	GEMM	0.065s	0.002s	28x
Honor 10	Overall	0.142s	0.018s	7.7x
	GEMM	0.131s	0.007s	17.4x

5 CONCLUSION

In this paper, we extend the prior work on systematic DNN weight pruning using ADMM. We integrate ADMM regularization with masked mapping/retraining, thereby guaranteeing solution feasibility and providing high solution quality. We develop two new applications: facial emotion detection and eye tracking, and propose a top-down framework of DNN training, model compression, and acceleration in mobile devices. The proposed method shows significant storage/memory reduction and speedup measured on mobile devices.

ACKNOWLEDGMENTS

This work is funded by the National Science Foundation Awards CNS-1704662 and CNS-1739748.

REFERENCES

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning* 3, 1 (2011), 1–122.

[2] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[3] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[4] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. 2017. NeST: a neural network synthesis tool based on a grow-and-prune paradigm. *arXiv preprint arXiv:1711.02017* (2017).

[5] Gene H Golub and Charles F Van Loan. 2012. *Matrix computations.* Vol. 3. JHU press.

[6] Ian J Goodfellow, Dumitru Erhan, Pierre Luc Carrier, Aaron Courville, Mehdi Mirza, Ben Hamner, Will Cukierski, Yichuan Tang, David Thaler, Dong-Hyun Lee, et al. 2013. Challenges in representation learning: A report on three machine learning contests. In *International Conference on Neural Information Processing.* Springer, 117–124.

[7] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org. (2010).

[8] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems.* 1379–1387.

[9] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[10] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems.* 1135–1143.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.

[12] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, Vol. 2.

[13] Jostine Ho. 2016. Facial Emotion Recognition. https://github.com/JostineHo/mememoji. (2016).

[14] Mingyi Hong, Zhi-Quan Luo, and Meisam Razaviyayn. 2016. Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM Journal on Optimization* 26, 1 (2016), 337–364.

[15] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. 2017. Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *arXiv preprint arXiv:1704.05519* (2017).

[16] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, and Antonio Torralba. 2016. Eye Tracking for Everyone. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems.* 1097–1105.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[19] Zhe Li, Xiaolong Ma, Hongjia Li, Qiyuan An, Aditya Singh Rathore, Qinru Qiu, Wenyao Xu, and Yanzhi Wang. 2018. C3PO: Database and Benchmark for Early-stage Malicious Activity Detection in 3D Printing. *arXiv preprint arXiv:1803.07544* (2018).

[20] Sheng Lin, Ning Liu, Mahdi Nazemi, Hongjia Li, Caiwen Ding, Yanzhi Wang, and Massoud Pedram. 2018. FFT-based deep learning deployment in embedded systems. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 1045–1050.

[21] Sijia Liu, Jie Chen, Pin-Yu Chen, and Alfred O Hero. 2017. Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications. *arXiv preprint arXiv:1710.07804* (2017).

[22] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. 2017. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922* (2017).

[23] Hua Ouyang, Niao He, Long Tran, and Alexander Gray. 2013. Stochastic alternating direction method of multipliers. In *International Conference on Machine Learning.* 80–88.

[24] Taiji Suzuki. 2013. Dual averaging and proximal gradient descent for online alternating direction multiplier method. In *International Conference on Machine Learning.* 392–400.

[25] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. 2017. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2097–2106.

[26] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems.* 2074–2082.

[27] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic DNN weight pruning framework using alternating direction method of multipliers. *arXiv preprint arXiv:1804.03294* (2018).