Mobilenet Model Compression with ADMM Training and Mix-up

A Thesis Presented
by

Zhengang Li

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Master of Science

in

Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

Apr 2019

Long may the sun shine.

Contents

List of Figures

List of Tables

List of Acronyms

ADAM	deep neural network.
ADMM	Alternating Direction Method of Multipliers.
BN	Bench Normalization.
CNN	deep neural network.
COO	coordinate format.
CSR	compressed sparse row format.
CONV	convolution layer.
DNN	deep neural network.
FC	fully connected layer.
FPGA	Field Programmable Gate Arrays.
GPU	graphics processing unit.
SGD	deep neural network.
ReLU	rectified linear unit.

Acknowledgments

Here I wish to thank my advisor Yanzhi Wang and all of my teammates. Without their help, this work could not be finished.

Abstract of the Thesis

Mobilenet Model Compression with ADMM Training and Mix-up

by

Zhengang Li

Master of Science in Electrical and Computer Engineering

Northeastern University, Apr 2019

Yanzhi Wang, Adviser

Weight pruning is a representative DNN model compression technique to simultaneously reduce the storage requirement and accelerate inference computation, with minor accuracy loss. There are two ways of pruning: structured and non-structured. With the former one, arbitrary weight in the neural network can be pruned. With the latter one, we prune the weight in the same channel, column or filter to bring a better compatibility in GPU.

MobileNet V2 with inverted residuals and linear bottleneck is built in 2018 which improves the state of the art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. It has a small size which makes it suitable for mobile devices.

In this paper, we combine pruning and MobileNet together to bring out models with extreme small size and high accuracy based on Cifar10. For realizing this purpose, we also add in a lot of different methods like ADMM training, mix-up training, progressive pruning, label smoothing, learning rate warm up, cosine scheduler and so on. With all of things above, we compare different pruning methods, discuss the unique features which expressed in the MobileNet pruning and bring out an error correction model for pruning redundancy. We also bring out several future works' direction based on experiments' discovery.

Chapter 1

Introduction

Recently, *model compression* techniques for deep neural networks (DNNs) have been intensively studied. These techniques, which are applied during the training phase of the DNN, aim to simultaneously reduce the model size (thus, the storage requirement) and accelerate the computation for inference with minor classification accuracy loss. [36, 8, 5, 21, 7, 35, 30, 13, 34, 26, 6, 27, 11, 38, 33, 3, 41, 42, 1, 2]. Two important categories of compression techniques are *weight pruning* and *weight quantization*.

For weight pruning, the early work focused on heuristic and iterative methods[15].

This weight pruning method has been extended in [10, 40, 14, 12] using more sophisticated heuristics. As a result, they are less compatible with the highly data parallel execution model in GPUs and multicore CPUs. Because the model after pruning does not have a size change. People can only use compressed matrix to make a post processing. To overcome the limitation of the non-structured weight pruning, recent works [37, 17] proposed to incorporate *regularity or "structures" in weight pruning*, including filter pruning, channel pruning, and filter shape pruning. The weight matrix will maintain a full matrix but with reduced dimensions, and indices are no longer needed. As a result, it leads to much higher speedups in GPUs.

Weight quantization is an orthogonal compression technique and has also been investigated [22, 28, 45, 23, 39, 29, 20, 9]. Compared to weight pruning, weight quantization is naturally more hardware-friendly, since both storage and computation of DNNs will be reduced proportionally to the weight precision with usually tolerable accuracy loss. As a result, weight quantization is becoming a "must-try" step for DNN inference engines. Later works realized a higher accuracy for compressed model. However, most of their works are based on AlexNet, VGG, ResNet, these huge networks, and lack the algorithm-wise guarantee of solution feasibility.

To overcome these limitations, we choose MobileNet as our model to compress, and add ADMM train into the pruning and quantization to promise all the constraints are satisfied. With MobileNet's separable convolution layers which has high compatibility in mobile devices, the total computation number can decrease in a further step. In this process, we look for the best combination of compression techniques to bring MobileNet a better inference capability. There are a series of comparison experiments in this work to guarantee the compressed models' quality. Our framework is built on CIFAR-10 data sets. For more specific information, please refer to the chapter 5. The best compressed model that we compress can be 95.38% for Cifar-10 accuracy.

Our second contribution is to raise a redundancy make-up model to deal with unused weights' error. The motivation of this part of work comes from the structured pruning in MobileNet V2. Because of MobileNet V2's special structure, it express a lot of different features in compression compared with other large networks, which have a huge influence at the expression of inference. To get rid of the influence comes from BN and ReLu layers, we make these mathematical derivations to restore the pruned model and help it to keep full accuracy after removing the redundancy weights.

Chapter 2

Model compression methods and ADMM training

2.1 Concept of Weight Pruning

Weight pruning can be defined into two styles: structured pruning and non-structured pruning. The latter one was brought out by Han. [15] In his work, he achieved 9× reduction in the number of parameters in AlexNet and 13× in VGG-16. Although most reduction is achieved in FC layers and this will not work in GPU acceleration, It is still a master piece to come up with a brand new thought of network compression. In the later work, like [14] (dynamic network surgery) and [10] (NeST), they use different ways to select weight being pruned, and get higher pruning rate in CONV layers in AlexNet. But the pruning rate is still kept in 3 to 4 times. Considering the AlexNet is a huge network, the compressed model is also with large size. In non-structured pruning, even if the main pruning part is at CONV layers, it is still hard to accelerate the GPU computation speed in direct using. Because ”zeros” in the neural network is distributed in irregular position. The computations of them are kept in GPU if we do not give a compressed sparse model such as CSR and COO.

To overcome the limitation in non-structured, People comes up with structured pruning which is also known as regular pruning. In the work of SSL [37], the pruning part is separated by conception of filter, channel and so on which is focused on CONV layers. With these, we can shrink the size of CONV layers with regular size (”zeros” part are accumulated in the same filters or channels). In this way, the calculation size in GPU is decreased directly and the speed of

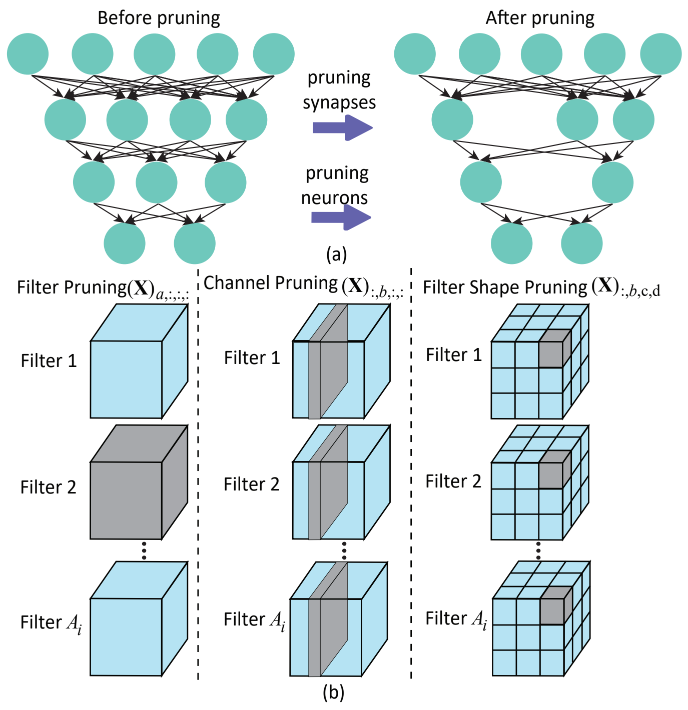

Before pruning
After pruning
pruning synapses
pruning neurons
(a)
Filter Pruning$(\mathbf{X})_{a,:,:,:}$
Channel Pruning $(\mathbf{X})_{:,b,:,:}$
Filter Shape Pruning $(\mathbf{X})_{:,b,c,d}$
Filter 1
Filter 2
Filter A_i
(b)

Figure 2.1: (a) Non-structured weight pruning (arbitrary weight can be pruned) and (b) three types of structured weight pruning.

GPU computation is accelerated. However, the weight pruning rate is limited in the prior work on structured pruning. However, in regular pruning works, the pruning rate is hard to be increased. We can only get 1.4 times in this work at Alexnet which gives out a compressed model with a large size.

2.2 Three Types of Structured Pruning

In Wen *et al.* [37]'s work, there are three types of structured pruning: *filter pruning*, *channel pruning*, and *filter shape pruning*, as shown in blackFigure 2.1 (b). Filter pruning removes whole filters; channel pruning removes whole channels; and filter shape pruning removes the weights in the same locations of all filters in one specific layer.

Moreover, as shown in blackFigure 2.2 and as shown in blackFigure 2.3, structured pruning has some feature of corresponding influence between layers by layers. Every pruned filter has a corresponding channel in the next layer, and every pruned channel has a corresponding filter in the last layer. This is because that filter or channel pruning can produce unused feature map which results in unused filter or channel in the corresponding layers. As a result, filter pruning (and channel

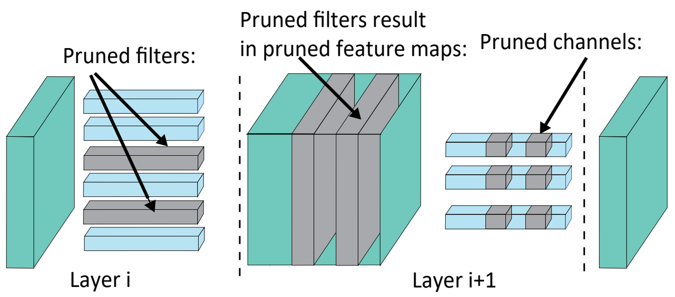


Figure 2.2: Relation between filter pruning and channel pruning.

pruning) has a roughly quadratic effect on the reduction in the number of weight parameters (and the amount of computations) of the DNN. Specifically, to a DNN without bias and BN layers, Pruning a filter in layer i is equivalent to pruning the corresponding channel in layer $i + 1$, which is generated by this specific filter. Unfortunately, most DNNs have bias or BN layers to get a better accuracy. So, with different neural network having different expression, these corresponding redundancy weights can not be removed in that easy way. In this thesis, I will illustrate this part in detail and give a calculation method of redundancy makeup.

Filter shape pruning has different version to realize. Generally, we can separate it as column pruning and pattern pruning. In blackFigure 2.3 (b), we can get a brief concept of column pruning. Also, when a channel's all the columns are pruned, the channel is empty (something like channel pruning). In this situation, we can still have redundancy weights in the last layers like channel pruning.

With the using of structured pruning, we can reduce the dimensions in weight matrix while maintaining a full matrix format. Thus, indices which CSR or other compressed matrices use in non-structured pruning are not needed anymore. It is why structured pruning techniques are more suitable for hardware accelerations in general.

In conclusion, the major advantage of structured pruning is the superlinear effect on storage/computation reduction. In our work, taking relation between layers by layers into consider, structured pruning can sometimes have higher compressed ratio than non-structured pruning.

2.3 Weight Quantization

Weight quantization. This method takes advantages of the inherent redundancy in the number of bits for weight representation. We can use less bits to express each weight and get a considerable storage and computation reduction. Many of the prior works [22, 28, 45, 23, 39, 29, 20, 9, 9, 22, 22] focused on quantization of weights with acceptable accuracy loss. Generally, the quantized weight can binary, ternary or powers of 2 values. From now on weight quantization is becoming a must-try step in DNN compression and it is supported not only in GPU, but also in FPGA and mobile devices.

In our work, we also choose several pruned model with high quality to quantize in different bits and get impressive results.

2.4 ADMM for Weight Pruning/Quantization

ADMM is a powerful tool for optimization. With this, the accuracy after compreesion can be increased a lot. Recent work [44, 22] have incorporated ADMM for DNN weight pruning and weight quantization, respectively. The main thought is to decompose the original problem into two subproblems that can be solved separately and efficiently. For example, considering optimization problem $\min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{x})$. In ADMM, this problem is decomposed into two subproblems on $\mathbf{x}$ and $\mathbf{z}$ (auxiliary variable), which will be solved iteratively until convergence. The first subproblem derives $\mathbf{x}$ given $\mathbf{z}$: $\min_{\mathbf{x}} f(\mathbf{x}) + q_1(\mathbf{x}|\mathbf{z})$. The second subproblem derives $\mathbf{z}$ given $\mathbf{x}$: $\min_{\mathbf{z}} g(\mathbf{z}) + q_2(\mathbf{z}|\mathbf{x})$.

However, due to the non-convex nature of the objective function for DNN training, there is still a lack of guarantee in the prior work on solution quality. Part of prior work even have trouble at the combination of ADMM with structured pruning. In our work, we build a set of projects in both non-structured and structured weight pruning and combine a series of techniques to enhance the model which guarantee solution feasibility and provide high solution quality.

2.4.1 Problem Formulation

Consider an N-layer DNN with both CONV and FC layers. The weights and biases of the i-th layer are respectively denoted by $\mathbf{W}_i$ and $\mathbf{b}_i$, and the loss function associated with the DNN is denoted by $f(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N)$; see [44]. In this paper, $\{\mathbf{W}_i\}_{i=1}^N$ and $\{\mathbf{b}_i\}_{i=1}^N$ respectively characterize the collection of weights and biases from layer 1 to layer N. Then DNN weight pruning or weight quantization is formulated as the following optimization problem:

$$\{\mathbf{W}_i\}, \{\mathbf{b}_i\} minimize f(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N), subject to \mathbf{W}_i \in \mathcal{S}_i,\ i = 1, \ldots, N, \tag{2.1}$$

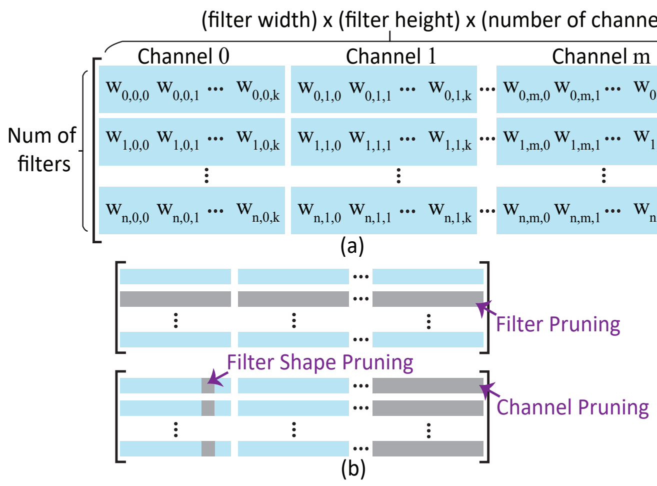


Figure 2.3: (a) To support GEMM computation, the weight tensor representation of a CONV layer is transformed into the weight matrix representation. (b) How different structured weight pruning schemes are implemented on the weight matrix representation.

Next we introduce constraint sets $\mathcal{S}_i$'s corresponding to the general, non-structured weight pruning, different types of structured pruning, as well as weight quantization. We use CONV layers as illustrative example since CONV layers are the most computationally intensive. But the problem formulation can be well applied to FC layers [44].

The collection of weights in the i-th CONV layer is a four-dimensional tensor, i.e., $\mathbf{W}_i \in R^{A_i \times B_i \times C_i \times D_i}$, where A_i, B_i, C_i, and D_i are respectively the number of filters, the number of channels in a filter, the height of the filter, and the width of the filter, in layer i. In the following part, if $\mathbf{X}$ denotes the weight tensor in a specific layer, let $(\mathbf{X})_{a,:,:,:}$ denote the a-th filter in $\mathbf{X}$, $(\mathbf{X})_{:,b,:,:}$ denote the b-th channel, and $(\mathbf{X})_{:,b,c,d}$ denote the collection of weights located at position $(:, b, c, d)$ in every filter of $\mathbf{X}$, as illustrated in blackFigure 2.1 (b).

Weight pruning: For the *general, non-structured weight pruning*, the constraint on the weights in i-th layer is $\mathbf{W}_i \in \mathcal{S}_i := \{\mathbf{X} \mid$ number of nonzero elements in $\mathbf{X}$ is less than or equal to $\alpha_i\}$. For *filter pruning* (row pruning), the constraint in the i-th CONV layer becomes $\mathbf{W}_i \in \mathcal{S}_i := \{\mathbf{X} \mid$ the number of nonzero filters in $\mathbf{X}$ is less than or equal to $\beta_i\}$. For *channel pruning*, the constraint becomes $\mathbf{W}_i \in \mathcal{S}_i := \{\mathbf{X} \mid$ the number of nonzero channels in $\mathbf{X}$ is less than or equal to $\gamma_i\}$. Finally, for *filter-shape pruning* (column pruning), the constraint in the i-th CONV layer is $\mathbf{W}_i \in \mathcal{S}_i := \{\mathbf{X} \mid$ the number of nonzero vectors in $\{\mathbf{X}_{:,b,c,d}\}_{b,c,d=1}^{B_i,C_i,D_i}$ is less than or equal to $\theta_i\}$. These α_i, β_i, γ_i, and θ_i values are hyperparameters determined in prior, and the determination procedure will be discussed in Section 2.4.4.

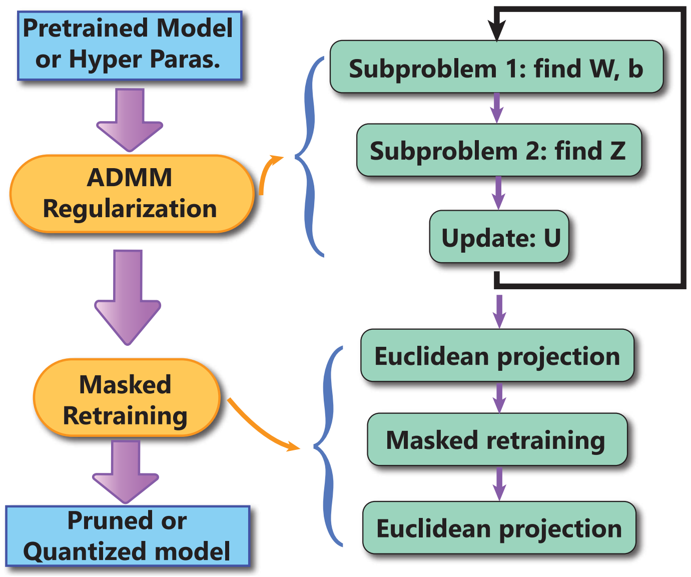


Figure 2.4: Procedure of ADMM-based unified weight pruning/quantization framework.

Weight quantization: For weight quantization, elements in $\mathbf{W}_i$ assume one of $q_{i,1}, q_{i,2}, \ldots, q_{i,M_i}$ values, where M_i denotes the number of these fixed values. Here, the $q_{i,j}$ values are *quantization levels* of weights of layer i in increasing order, and we focus on *equal-distance quantization* (the same distance between adjacent quantization levels) to facilitate hardware implementation.

2.4.2 Enhancing the ADMM-based Framework

In problem (2.1), the constraint is combinatorial. As a result, this problem cannot be solved directly by stochastic gradient descent methods like original DNN training. However, the form of the combinatorial constraints on $\mathbf{W}_i$ is compatible with ADMM which is recently shown to be an effective method to deal with such clustering-like constraints [18, 24].

Despite such compatibility, it is still challenging to directly apply ADMM due to the non-convexity in objective function. To overcome this challenge, we extend over [44, 22, 31] and develop a systematic framework of dynamic ADMM regularization, masked mapping and retraining steps. We can guarantee solution feasibility (satisfying all constraints) and provide high solution quality (pruning/quantization rate under the same accuracy) through this integration. This framework is unified and applies to both non-structured and structured weight pruning and weight quantization, see Figure 2.4.

ADMM Regularization Step: The ADMM regularization decomposes the original problem (2.1) into two subproblems. We omit the details due to space limitation. (i) defining indicator function $g_i(\mathbf{W}_i) = \begin{cases} 0 & if \mathbf{W}_i \in S_i \\ +\infty & otherwise \end{cases}$ corresponding to every set $\mathcal{S}_i$; (ii) incorporating auxiliary variables $\mathbf{Z}_i$, $i = 1, \dots, N$; and (iii) adopting augmented Lagrangian [4]. These decomposed subproblems will be iteratively solved until convergence. The first subproblem is

$$\{\mathbf{W}_i\}, \{\mathbf{b}_i\} minimize \quad f(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N) + \sum_{i=1}^N \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i^k + \mathbf{U}_i^k\|_F^2, \tag{2.2}$$

where $\mathbf{U}_i^k := \mathbf{U}_i^{k-1} + \mathbf{W}_i^k - \mathbf{Z}_i^k$. The first term in the objective function of (2.2) is the differentiable loss function of the DNN, and the second term is a quadratic regularization term of the $\mathbf{W}_i$'s, which is differentiable and convex. As a result (2.2) can be solved by stochastic gradient descent as original DNN training. Please note that this first subproblem maintains the same form and solution for (non-structured and structured) weight pruning and quantization problems.

On the other hand, the second subproblem is given by

$$\{\mathbf{Z}_i\} minimize \quad \sum_{i=1}^N g_i(\mathbf{Z}_i) + \sum_{i=1}^N \frac{\rho_i}{2} \|\mathbf{W}_i^{k+1} - \mathbf{Z}_i + \mathbf{U}_i^k\|_F^2. \tag{2.3}$$

Note that $g_i(\cdot)$ is the indicator function of $\mathcal{S}_i$, thus this subproblem can be solved analytically and optimally [4]. For $i = 1, \dots, N$, the optimal solution is the Euclidean projection of $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ onto $\mathcal{S}_i$. For *non-structured weight pruning*, 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. For *filter pruning*, we first calculate $O_a = \|(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{a,:,:,:}\|_F^2$ for $a = 1, \dots, A_i$, where $\|\cdot\|_F$ denotes the Frobenius norm. We then keep β_i elements in $(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{a,:,:,:}$ corresponding to the β_i largest values in $\{O_a\}_{a=1}^{A_i}$ and set the rest to zero. For *channel pruning*, we first calculate $O_b = \|(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{:,b,:,:}\|_F^2$ for $b = 1, \dots, B_i$. We then keep γ_i elements in $(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{:,b,:,:}$ corresponding to the γ_i largest values in $\{O_b\}_{b=1}^{B_i}$ and set the rest to zero. The optimal solution of the second subproblem for *filter shape pruning* is similar, and is omitted due to space limitation. For *weight quantization*, we can prove that the Euclidean projection results in mapping every element of $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ to the quantization level closest to that element.

After both subproblems solved, we update the dual variables $\mathbf{U}_i$'s according to the ADMM rule [4] and thereby complete one iteration in ADMM regularization. Overall the ADMM regularization step can be understood as a smart, dynamic L_2 regularization, in which the regularization target $\mathbf{Z}_i^k - \mathbf{U}_i^k$ will change judiciously and analytically in each iteration. On the other hand, conventional regularization methods (based on L_1, L_2 norms or their combinations) use a fixed regularization target, and the penalty is applied on all the weights. This will inevitably cause accuracy degradation. Sample comparison results are provided in Section **??**.

Masked mapping and retraining: After ADMM regularization, we obtain intermediate $\mathbf{W}_i$ solutions. The subsequent step of masked mapping and retraining will guarantee the solution feasibility and improve solution quality. For non-structured and structured weight pruning, the procedure is more straightforward. We first perform the said Euclidean projection (mapping) to guarantee that pruning constraints are satisfied. 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 (solution quality) can be (partially) restored, and solution feasibility (constraints) will be maintained.

For weight quantization, the procedure is more complicated. The reason is that the retraining process will affect the quantization results, thereby solution feasibility. To deal with this issue, we first perform Euclidean projection (mapping) of weights that are close enough (defined by a threshold value ϵ) to nearby quantization levels. Then we perform retraining on the remaining, unquantized weights (with quantized weights fixed) for accuracy improvement. Finally we perform Euclidean mapping on the remaining weights as well. In this way the solution feasibility will be guaranteed.

2.4.3 Techniques for Enhancing Convergence

In this section we discuss two techniques for enhancing convergence (rate and results): multi-rho method in ADMM regularization, and progressive weight pruning. We abandon the extragradient descent method in [22] as we did not find the advantage in convergence speed, not to mention the additional hyperparameters introduced by this method.

Increasing ρ in ADMM regularization: The ρ_i values are the most critical hyperparameter in ADMM regularization. We start from smaller ρ_i values, say $\rho_1 = \ldots = \rho_N = 1.5 \times 10^{-3}$, and gradually increase with ADMM iterations. This coincides with the theory of ADMM convergence [18, 24]. It in general takes 8 - 12 ADMM iterations for convergence, corresponding to 100 - 150 epochs in PyTorch. This convergence rate is comparable with the original DNN training.

Progressive weight pruning: The ADMM regularization is L_2 regularization. As a result, there is a large portion of very small weights values after one round of ADMM-based (non-structured or structured) weight pruning. This gives rise the opportunity to perform a second round of weight pruning. In practice, we perform *two rounds* of ADMM-based weight pruning consecutively, where the weight pruning results in the first round will be the starting point of the second round (weights that are already pruned to zero will not be recovered). This method has an additional benefit of reducing the search space in each step, thereby accelerating convergence.

2.4.4 Hyperparameter Determination

Hyperparameter determination mainly refers to the determination process of pruning rate (e.g., the α_i value) and/or the number of quantization levels per layer of DNN. This is a more challenging task for pruning than quantization in general. For quantization, it is typically preferred for the same number of quantization levels for all (or most of) layers, like binarized or ternarized weights, which is preferred by hardware. For weight pruning, on the other hand, these pruning rate values are flexible and shall be judiciously determined.

As hyperparameter determination is not our primary focus, we use a heuristic method as follows. We observe that we can achieve at least $3\times$ more weight pruning than prior, heuristic weight pruning methods without accuracy loss. Hence, we adopt the per-layer pruning rates reported in prior work, and increase proportionally. In the progressive pruning procedure, we set the target of the first round to be $1.5\times$ pruning than prior work, and the second round to be doubled based on that. We will further increase the pruning rates if there is still margin for weight pruning without accuracy loss.

Chapter 3

Introduction of MobileNet

3.1 Previous Work of MobileNet

MobileNet is an architecture of neural network which is more suitable for mobile and embedded based vision applications where there is lack of compute power. This architecture was proposed by Google.

From now on, MobileNet has two version, MobileNet V1[19] reap the benefits of Xception, giving a small-size model with high accuracy. This architecture uses separable convolutions consisting of depthwise convolution and pointwise convolution (in figure 3.1), which significantly reduces the number of parameters when compared to the network with normal convolutions with the same depth in the networks.

In the normal convolution, if the filter's kernel size is of [C x L x L] dimension and there

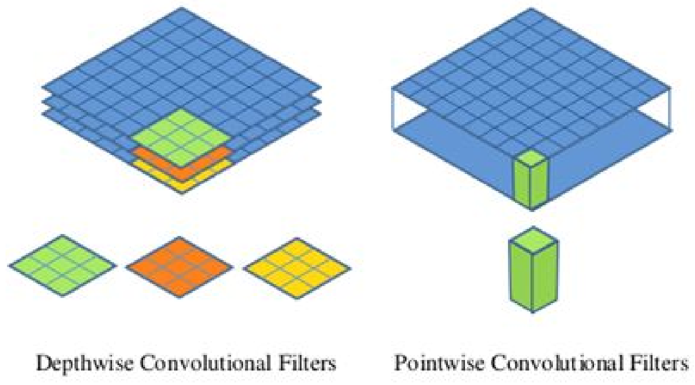

Depthwise Convolutional Filters
Pointwise Convolutional Filters

Figure 3.1: two layers in separable convolution

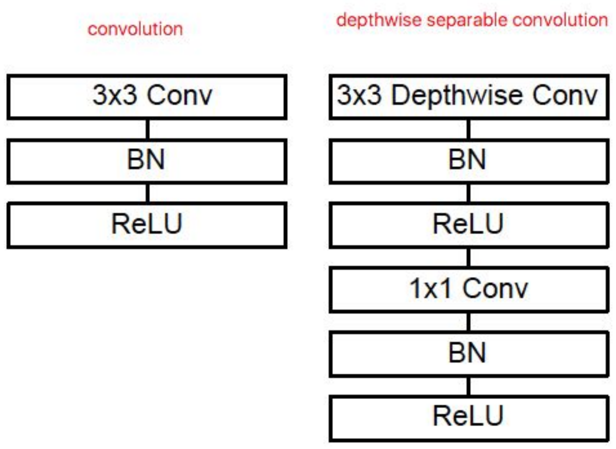


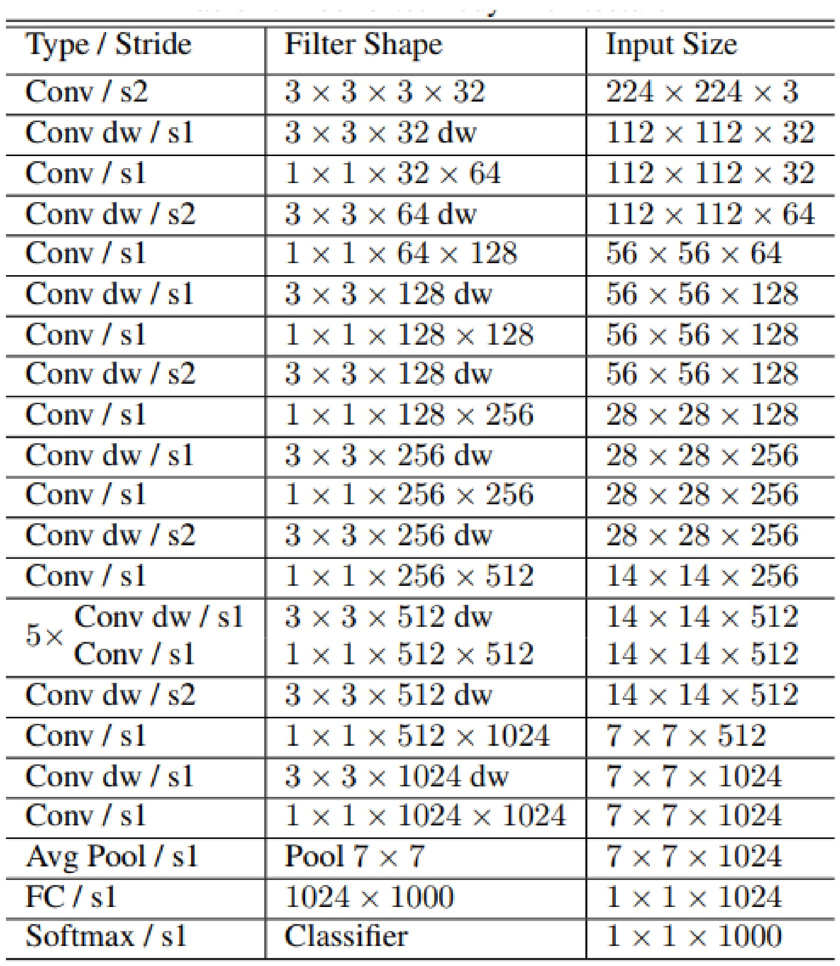
Figure 3.2: Normal Convolution and Separable Convolution

are k filters. For computing one pixel's all channels in the next layer's input feature map, normal convolution needs [C x L x L x k] times multiplication. But if we use separable convolution, we only need [C x L x L] multiplications in depthwise layer and [C x k] multiplications in pointwise layer. Even if MobileNet has the same size as normal neural networks, the total computation complexity decreases a lot.

MobileNet V1 is impressive, but there is still some sacrifice of accuracy. To overcome this limitation, work in MobileNet V2 [32] combines inverted residuals and linear bottlenecks to get a smaller and more high-accuracy model compared with MobileNet V1. Tensorflow official website offers the 100% depth MobileNet V2 model with 91.0% Top 5 accuracy in ImageNet dataset.

3.2 Structrue of MobileNet

MobileNet V1 is based on a streamlined architecture which uses depthwise separable convolutions to build light weight deep neural networks.[19] Each block in V1 is like figure 3.2. MobileNet V1 has global hyper-parameters used to trade off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem. To be specific, the structural integrity can be defined by user. In our work, we only choose network with 100% integrity to compress to get the full sight. This structure can be expressed as figure 3.3. The ”Conv dw” in the form means depthwise convolution



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s1	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 3.3: structure of mobilenet V1 in ImageNet dataset

layer, every ”Conv” after a depthwise layer is a corresponding pointwise layer. There are 13 blocks of separable convolutions in this model. The specific structure of each layer may change a little among different dataset.

MobileNetV2 builds upon the ideas from MobileNetV1, using depthwise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture: linear bottlenecks between the layers, and shortcut connections between the bottlenecks.The basic structure is shown as figure 3.4. This basic block contain three parts: the first pointwise layer to increase dimension, the second depthwise layer to reap features channel by channel, and the third pointwise layer to decrease dimension. The dimension expansion parameters are 1 or 6 in most cases. Besides that, shortcuts’ distribution follows two principle: 1) the stride of the correspond-

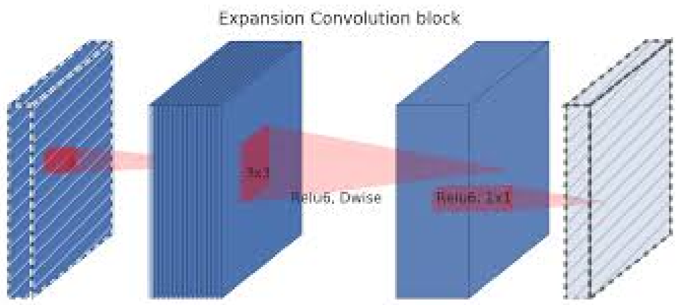


Figure 3.4: basic block in MobileNet V2

ing blocks should be 1; 2) the channel number of input and output should be different(in figure 3.5). The shortcut here is also a pointwise convolution with BN layer. It is worth stressing that, the shortcut appeared in MobileNet is different from ResNet. After the adding of shortcut and major layer, there is no ReLu. It is because that calculation of nonlinearity can impair the feature delivery in lower dimension.[32] This characteristic protect the feature integrity in the MobileNet but bring some difficulty to the removal of model compression redundancy. This part will be illustrated in Chapter 6.

Every bottleneck structure may have several blocks depending on the structure integrity. Figure 3.6 gives the overview of 100% structure-integrity MobileNet V2 in ImageNet dataset. In this structure, we have 7 bottlenecks with totally 17 blocks distributed in them. Three blocks (four blocks for Cifar10 dataset) have their corresponding shortcuts.

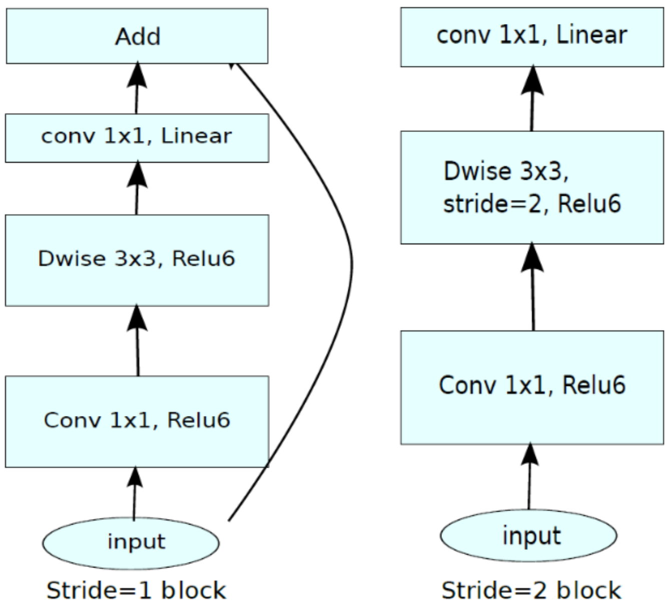
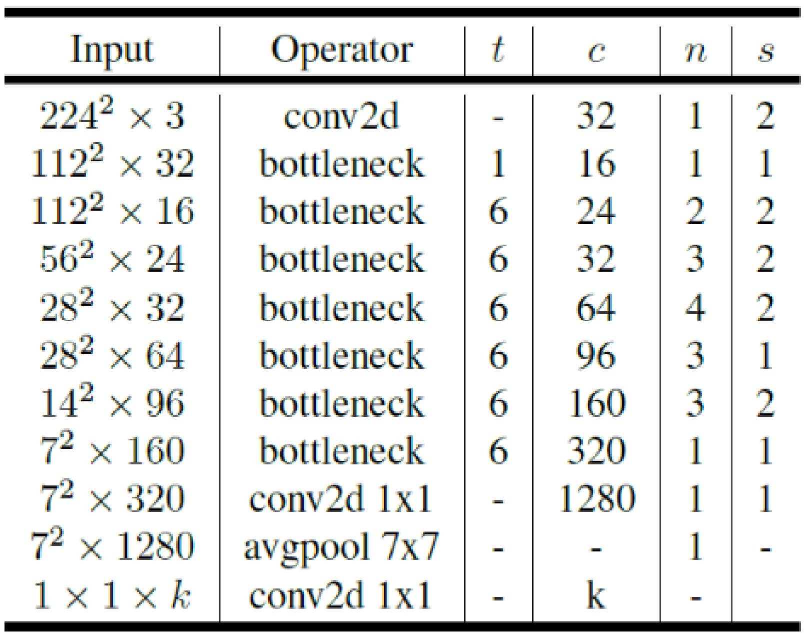


Figure 3.5: MobileNet V2 block's different structure



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$28^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times k$	conv2d 1x1	-	k	-	

Figure 3.6: Structure of MobileNet V2 in ImageNet dataset

Chapter 4

Introduction of Mix-up and Other Used Methods

In this chapter, we choose part of works which have high compatibility with ADMM training from Bag of Tricks for Image Classification with Convolutional Neural Network[16]. With these methods, the accuracy of mobilenet can be improved by two percentage whatever in baseline or in pruned and quantized model.

4.1 Mix-up Training

The conception of Mix-up is first raised in "mixup: Beyond Empirical Risk Minimization"[43]. The mixup vicinal distribution can be understood as a form of data augmentation that encourages the model to behave linearly in-between training examples. In mixup, each time we randomly sample two examples ($\mathbf{x}_i$, $\mathbf{y}_i$) and ($\mathbf{y}_j$, $\mathbf{y}_j$). Then we form a new example by a weighted linear interpolation of these two examples:

$$\begin{aligned}\hat{x} &= \lambda x_i + (1-\lambda)x_j \\ \hat{y} &= \lambda y_i + (1-\lambda)y_j\end{aligned} \tag{4.1}$$

where $\lambda \in$[0, 1] is a random number drawn from the Beta(α, α) distribution. In mixup training, we only use the new example ($\hat{x}$, $\hat{y}$).

4.2 Learning Rate Warm-up

At the beginning of the training, all parameters are typically random values and therefore far away from the final solution. Using a too large learning rate may result in numerical instability. In the warmup heuristic, we use a small learning rate at the beginning and then switch back to the initial learning rate when the training process is stable. Previous work proposes a gradual warmup strategy that increases the learning rate from 0 to the initial learning rate linearly. In other words, assume we will use the first m batches (e.g. 5 data epochs) to warm up, and the initial learning rate is k, then at batch i which is in the range of 1 to m, we will set the learning rate to be ik/m [43]

4.3 Label Smoothing

The last layer of a image classification network is often a fully-connected layer with a hidden size being equal to the number of labels, denote by K, to output the predicted confidence scores. Given an image, denote by zi the predicted score for class i. These scores can be normalized by the softmax operator to obtain predicted probabilities. Denote by q the output of the softmax operator q = softmax(z), the probability for class i, qi , can be computed by:

$$q_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)} \tag{4.2}$$

Its easy to see qi is larger than 0 and PK i=1 qi = 1, so q is a valid probability distribution. On the other hand, assume the true label of this image is y, we can construct a truth probability distribution to be pi = 1 if i = y and 0 otherwise. During training, we minimize the negative cross entropy loss

$$\ell(p,q) = -\sum_{i=1}^{K} q_i \log p_i \tag{4.3}$$

to update model parameters to make these two probability distributions similar to each other. It encourages the output scores dramatically distinctive which potentially leads to overfitting. The idea of label smoothing was first proposed to train Inception-v2. It changes the construction of the true probability to

$$q_i = \begin{cases} 1-\varepsilon & if, i = y \\ \varepsilon/(K-1) & otherwise \end{cases} \tag{4.4}$$

where is a small constant. Now the optimal solution becomes

$$z_i^* = \begin{cases} \log((K-1)(1-\varepsilon)/\varepsilon) + \alpha & if, i = y \\ \alpha & otherwise \end{cases} \tag{4.5}$$

where can be an arbitrary real number. This encourages a finite output from the fully-connected layer and can generalize better.

It is clear that with label smoothing the distribution centers at the theoretical value and has fewer extreme values.

4.4 Cosine Learning Rate Scheduler

Learning rate adjustment is crucial to the training. After the learning rate warmup described above, we typically steadily decrease the value from the initial learning rate. The widely used strategy is exponentially decaying the learning rate. In general work, people always use learning rate adjustment as ladder-like, to decrease the learning rate after a certain number of epoches. In contrast to it, Loshchilov [25] propose a cosine annealing strategy. An simplified version is decreasing the learning rate from the initial value to 0 by following the cosine function. Assume the total number of batches is T (the warmup stage is ignored), then at batch t, the learning rate η_t is computed as:

$$\eta_t = \frac{1}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)\eta \tag{4.6}$$

where is the initial learning rate. We call this scheduling as cosine decay.

The cosine decay decreases the learning rate slowly at the beginning, and then becomes almost linear decreasing in the middle, and slows down again at the end. Compared to the step decay, the cosine decay starts to decay the learning since the beginning but remains large until step decay reduces the learning rate by 10 times, which potentially improves the training progress. [43]

Chapter 5

Experiment Result and Analysis

In this chapter, we bring out a series of compressed model based on MobileNet. Because the methods we have mentioned in Chapter 4 need much more time in training(nearly 200 more epochs in each training step). Before applying them, we use basic training techniques and ADMM pruning to compare the compressibility of MobileNets with two versions and choose the better one to continue with full compression techniques.

All the pruned rates and total compression rates in this chapter are for convolution layers.

5.1 Comparison of pruning result between MobileNet V1 with MobileNet V2 without Mix-up

In this section, we give equal pruning ratio to models' each layer. Increasing the pruning ratio step by step and stopping when the accuracy start to decrease obviously, we get pruned models as table 5.1 and table 5.2

pruning version	prune rate	accuracy	accuracy change
baseline	1.00×	91.31%	–
non-structured	2.00×	91.88%	+0.57%
non-structured	3.33×	91.00%	-0.31%
non-structured	5.00×	89.97%	-1.34%

Table 5.1: MobileNet V1 Non-structured Pruning without Mix-up Training in Cifar10

As what is shown here, we can find that MobileNet V2 has a better compressibility compared with V1. It is because that MobileNet V2 has bottleneck structure. The number of channels

pruning version	**prune rate**	**accuracy**	**accuracy change**
baseline	1.00×	93.30%	–
non-structured	2.00×	93.61%	+0.31%
non-structured	3.33×	93.41%	+0.11%
non-structured	5.00×	93.39%	+0.09%
non-structured	6.67×	92.85%	-0.45%
non-structured	10.00×	92.11%	-0.89%

Table 5.2: MobileNet V2 Non-structured Pruning without Mix-up Training in Cifar10

has been expanded before getting into depthwise layers. This feature makes the network more stable in the pruning process.

5.2 Comparison of different structured prune version in MobileNet V2 without mix-up

In this section, we are going to find a proper method of structured pruning in MobileNet V2 before applying full techniques. Because MobileNet V2 has bottleneck structure which contains 1×1 ponitwise convolution layer, so channel prune is equal to column prune in MobileNet V2. Here, we compare the accuracy after column pruning with filter pruning and increase the prune ratio step by step until an obvious decrease in accuracy occurs. In the table 5.3, we can see that, with the same

pruning version	**filter pruned**	**accuracy**	**accuracy change**
baseline	0	93.30%	–
filter prune	50%	93.32%	+0.02%
filter prune	60%	93.16%	-0.14%
filter prune	65%	92.75%	-0.55%
column prune	60%	91.73%	-1.27%
column prune	65%	90.50%	-2.80%

Table 5.3: MobileNet V2 structured Pruning with even prune rate in each layer on Cifar10

and even pruning ratio, model after filter prune has a better accuracy, which gives us a direction in later work.

5.3 Non-structured Pruning of MobileNet V2 with Mix-up Training

To keep the model consistency, we apply all the techniques into the baseline training, and get the baseline model with 95.37% accuracy for Cifar10 dataset. All the later works are based on this model. Table 5.4 gives the result of non-structured pruning of MobileNet V2 with mix-up, label smoothing, cosine scheduler and learning rate warm-up. The accuracy of baseline and models with specific prune rate are all increased by 2% synchronously compared with work in section 5.1.

pruning version	**prune rate**	**accuracy**	**accuracy change**
baseline	1.00×	95.37%	–
non-structured	3.33×	95.49%	+0.12%
non-structured	5.71×	95.04%	-0.33%
non-structured	6.67×	94.70%	-0.67%
non-structured	10.00×	93.75%	-1.62%

Table 5.4: MobileNet V2 Non-structured Pruning in Cifar10

5.4 Structured Pruning Result of MobileNet V2 with Mix-up Training

In structured pruning, the pruning ratio of each layer has a huge effect on the robust of model. With the different pruning ratio distribution, the accuracy and redundancy of model may have huge difference. For instance, if we use high filter prune ratio at layers with large weight size and use small filter prune ratio at layers with small weight size, the total pruned weights number can be larger with the same average filter pruned ratio. In this case, the redundant channel of corresponding pruned filter may also be huge, because broad layers are always beside each other.

In the experiments, we find that the structure of pointwise convolution and shortcut are suitable for filter pruning, but the depthwise convolution does not have a good compatibility with it. Fortunately, most of weights are allocated in depthwise layers in each block, so we can accumulate the pruning ratio from depthwise to pointwise for a better result. In table 5.3, we show part of the 60% filter prune rate result with different pruning ratio distribution. The first one is with even prune rate for each layer; the second one has more pruned filters in large layers and less pruned filters in small layers; the third one pushes part of pruning rate in depthwise layers into the corresponding pointwise layers; the forth one pushes all the pruning ratio from depthwise to pointwise.

We can see that, when we try to prune less depthwise and more pointwise layers, the pruned model accuracy goes higher. But if we overdo this, the accuracy will loss because of model's imbalance. So, it is important to find a new method for depthwise pruning. Here, we give a method which uses progressive training to combine pattern pruning and filter pruning together to work on MobileNet. With this, we use different pruning type in different layers and get higher accuracy.

pruning version	**prune distribution**	**accuracy**	**accuracy change**
baseline	–	95.37%	–
filter	even	94.88%	-0.49%
filter	uneven	95.12%	-0.25%
filter	uneven with less prune rate in depthwise	95.24%	-0.13%
filter	uneven with no prune rate in depthwise	94.96%	-0.41%
filter and pattern	uneven with pattern prune in depthwise and filter prune in pointwise	95.38%	+0.01%

Table 5.5: MobileNet V2 structured Pruning with 60% filter in Cifar10

Because all of them have pruning ratio accumulated at broad layers, so the total weight pruning after removing redundancy outdistance the prune rate shown in the table 5.5. In the next section, we will discuss the true compression ratio in the last pruned model.

5.5 Structured Pruning Redundancy analysis in MobileNet V2 and BN Pruning

In this section, we will talk about the redundancy after structured pruning and methods to wipe off them. Because of the special structure of MobileNet V2, we need to classify all the situations in filter pruning into five types:

1) Filter Pruning in the first pointwise layer of one block has redundancy channel in the second depthwise layer and third pointwise layer in the same block.

2) Additional filter pruning in the second layer has redundancy channel in the third depthwise layer in the same layer.

3) Filter pruning in the third pointwise layer has redundancy channel in the first pointwise layer of next block and parallel shortcut(if exist) .

4) If one block has a parallel shortcut, the redundancy channel of next block should be the intersection of the pruned filters of shortcut and the third pointwise layer.

5) The pruned filters in the last bottleneck have corresponding redundancy in the next normal convolution layer.

Here, we pick two typical situations to illustrate, the first type as figure 5.1 and the fourth type as figure 5.2

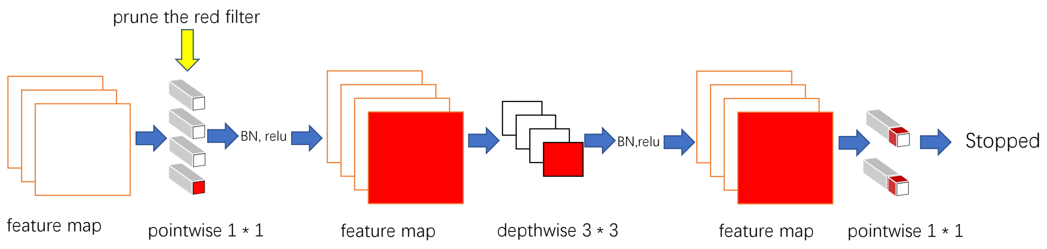


Figure 5.1: Redundancy comes from first pointwise layer in blocks

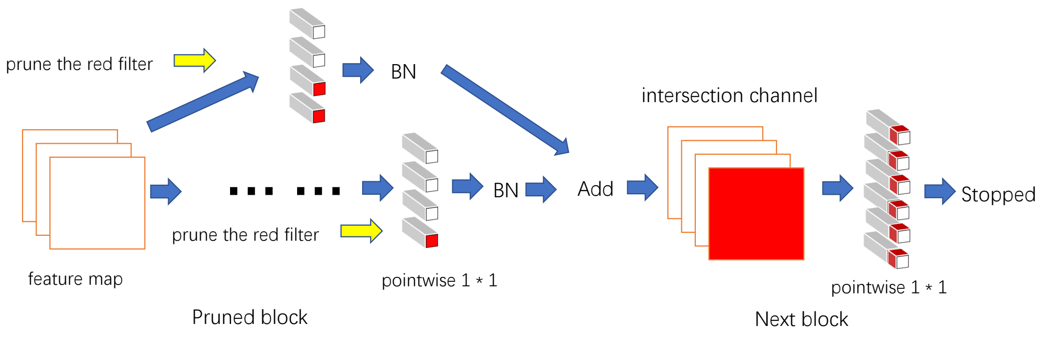


Figure 5.2: Redundancy comes from shortcut and third pointwise layer in blocks

In the first case, we have filter pruning at the first layer at one block. As what is shown in the picture, the red part is the redundancy which should be wiped off. This influence shall be shown both in second and third layers, because depthwise layer in MobileNet V2 has a "layer to layer" convolution. The corresponding unused channel is transmitted totally because of the zeros in the feature map. This redundancy transmission will end in the third layer with the redundancy only exists in part of each filter. In the second case, we have filter pruned both in shortcut and the last pointwise layer in the block. Because the feature map need to be added after these two parts' computation, the final feature map's zero part is only corresponding to the intersection of their pruned filters. This redundancy transmission only gets through one layer.

Taking everything into consider, the total CONV layers compression rate of the last model in figure 5.5 is 8.8825 $\times$, which is higher than the non-structured prune with the same accuracy.

From now on, everything seems to be OK, but there is still one problem: the BN layer that we apply after each convolution layer has a bias parameter which will make the feature map's corresponding channels not truly zero, but a non-zero constant. Wiping the corresponding channels in the net directly will result in an error of redundancy. This error has different influence in different network. If the network is large and with normal convolution like VGG-16 which we have tested, the error can sometimes be ignored.

But for MobileNet V2, a delicate network, cutting the corresponding layers roughly can make a huge accuracy loss. The reasons are mainly in two parts: 1) When we prune the first layer of each block, the redundancy can be transmitted through two layers which enlarge the error in two times; 2) When we prune the third layers in each block, the error will be transmitted completely into the next block because of no ReLu layers.

In experiments, we also try BN pruning to shrink the error. BN pruning is a special filter pruning which only prune the weight of BN layer. In this way we can decrease the error after BN layers from $\frac{-\mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}}\gamma+\beta$ to β. This can decrease the influence of the first part obviously, but has less effect at the second pat of error. Using BN pruning with the same "filter + pattern pruning" as the last pruned model, we can get model accuracy as 95.32% and get 95.28% model with only wiping out the first part of error. In this case, the pruning ratio can reach 4.975$\times$ without any makeup of redundancy.

5.6 Structured Pruning Redundancy Error Correction

In the section 5.5, we see that, BN pruning can not eliminate the redundancy error. So we need to find a proper way to correct the error. Here, we raise the redundancy error correction which adds a make-up parameter at redundant layers' BN bias. Every pruned filter has one or several independent make-up values to retune the output of BN layers. After this correction, the model after removing unused channel will have totally equal computation process as the original pruned model.

There are five situations in redundancy producing as mentioned in section 5.5. Every one of them has a specific make-up computation. The final result should be the summation of five make-ups. Due to limited space, here we only show the first situation with pruning the first layers.

First, we give a simple make-up for no padding layer:

In figure 5.1, we can see that, in the second feature map, the red channel which should be all zeros in ideal condition will has a constant $relu(\frac{-\mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}}\gamma + \beta)$ at every pixel. This is the first difference appearing in the redundancy transmission.

After the depthwise, if the network is no-padding, the output of one redundant filter should be:

$$F_{depthwise} = relu(\frac{-\mu_{\mathcal{B}1}}{\sqrt{\sigma_{\mathcal{B}1}^2 + \epsilon}}\gamma_1 + \beta_1) \tag{5.1}$$

After the BN and relu layers behind the depthwise layer, the corresponding channel of the output feature map should have the same value in each pixel as:

$$F_{feature2} = relu(\frac{W_2 F_{depthwise} - \mu_{\mathcal{B}2}}{\sqrt{\sigma_{\mathcal{B}2}^2 + \epsilon}}\gamma_2 + \beta_2) \tag{5.2}$$

Here, W_2 means the summation of the filters' all the weights.

In the third pointwise layer which is not channel to channel convolution, the error will be sent into every channel equally, one channel's BN make-up should be the sum of all filters' influence. For the i-th filter, the error in the j-th channel should be:

$$F_{error} = \frac{W_{3j} F_{feature2i} - \mu_{\mathcal{B}3j}}{\sqrt{\sigma_{\mathcal{B}3j}^2 + \epsilon}}\gamma_{3j} + \beta_{3j} \tag{5.3}$$

W_{3j} means the corresponding redundant weight in the j-th filter of the third pointwise layer.

If we directly remove the unused channel, the corresponding channel in the feature map before the last BN should be all zero. In this case, the j-th channel's output of the feature map after the last BN layer should be the output of non-pruned part adding a pruned part's additional value. The additional value should be:

$$F_{additional} = relu(\frac{-\mu_{\mathcal{B}3}}{\sqrt{\sigma^2_{\mathcal{B}3} + \epsilon}}\gamma_3 + \beta_3) \tag{5.4}$$

So that, the subtraction between F_{error} and $F_{additional}$ should be the make-up value:

$$F_{makeup} = \frac{W_{3j}F_{feature2i}}{\sqrt{\sigma^2_{\mathcal{B}3j} + \epsilon}}\gamma_{3j} \tag{5.5}$$

With all the pruned filter taking into consider, if there are N filters pruned in this case, the final make-up of the j-th channel's BN bias should be:

$$F_{makeupforj-thchannel} = \sum_{i=1}^{N} \frac{W_3 F_{feature2i}}{\sqrt{\sigma^2_{\mathcal{B}3j} + \epsilon}}\gamma_{3j} \tag{5.6}$$

After we finish all the types' make-ups and integrate them together, the original pruned model shall be restored totally for the no padding case. But, considering the padding layers, when we add additional zeros to the margin of feature map to make the map not shrunken after the computation, the corresponding channel of redundancy will not have the same value for all pixels. The value of 4 edges and 4 vertexes may have different value which will reflect in the W_2 in the computation of $F_{feature2}$. Here we can see the figure 5.2 to have an intuitive understanding.

In figure 5.3, the green part is the original feature map, small squares are the pointwise kernels' different location in computation. We can see that, at four margins and four vertexes which is outside the red line, part of the weight in the kernel will not be used. So that, the value of W_2 may only contains part of weights at specific position. For MobileNet V2, we only have 3×3 kernel in pointwise layers, so the make-up values for one channel should has 9 types. But in some networks which have larger kernal like 5×5, the classes of position should be more. With every two length size added in kernel, the total classes for one channel would be added by 8. This part we have not completed in programming. If possible, we may use the no-padding values to simulate the padding value to simplify the make-up computation.

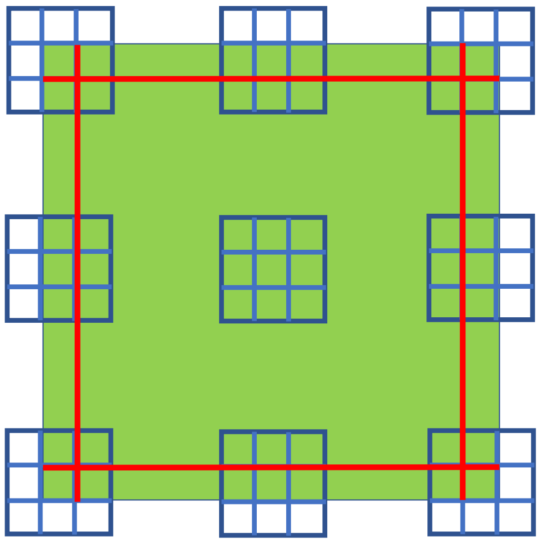


Figure 5.3: feature map in padding convolution

5.7 Final Result after Quantization

In this section, we choose 3 models which with high accuracy and high pruning ratio: 3.33× and 5.71× non-structured pruned model in table 5.4 and pattern/filter pruned model in table 5.5. Here, we use the full sparsity in structured pruned model to express the true compressed ratio(8.8825×).

pruned model	**quantized bits**	**total compression rate**	**accuracy**	**accuracy change**
non-structured 3.33×	6 bits	17.78×	95.48%	+0.11%
non-structured 3.33×	5 bits	21.33×	95.38%	+0.01%
non-structured 3.33×	4 bits	26.67×	95.17%	-0.20%
non-structured 5.71×	6 bits	30.45×	94.93%	-0.44%
non-structured 5.71×	5 bits	36.54×	94.86%	-0.51%
non-structured 5.71×	4 bits	45.68×	94.80%	-0.57%
pattern/filter 8.88×	6 bits	47.37×	95.16%	-0.21%
pattern/filter 8.88×	5 bits	56.85×	95.10%	-0.27%
pattern/filter 8.88×	4 bits	71.06×	95.09%	-0.28%

Table 5.6: MobileNet V2 Non-structured Pruning in Cifar10

Accuracy change in table 5.6 means the change between quantized model with baseline(95.37%). The total compressed rate in the table is contributed by both pruning and quantization.

With this final work. we can get compressed MobileNet V2 models with both high accuracy and compression ratio. As what has been mentioned in section 5.5 and 5.6, the last three extreme compressed models need redundancy makeup to realize the full quality. Without the makeup, the pruned rate can only be 4.9750$\times$ which will make the quantizied model's the total compression rate less than model based on "non-structured pruning 5.71$\times$" a little.

Chapter 6

Conclusion

Weight pruning is a representative DNN model compression technique to simultaneously reduce the storage requirement and accelerate computation for inference, with minor accuracy loss. With the quantization after the weight pruning, the model size can be compressed into an amazing size.

This paper gives a new idea to combine MobileNet with compression to bring out more efficient neural network, and compare the feature expressed in MobileNet V1 and V2 and choose suitable combination of compression methods to work on it.

To achieve this, we first use basic ADMM pruning framework that we build to compare the adaptability of two version of MobileNet and find that the MobileNet V2 has a better compressibility. Then, we use different pruning type on the MobileNet V2 to find the proper combination of pruning methods. After these basic work, we apply all the techniques like mix-up training to refine the work into best condition. At last, we apply quantization at the final pruned model to shrink the size with a further step.

After the work, we find that the redundancy of structure can not be got rid of directly due to the BN layers in MobileNet V2. So we trys BN pruning and use a series of deductive computation to get the make-up of redundancy. After this work, we can get the compressed model with extreme small size with full quality.

Bibliography

[1] `http://www.techradar.com/news/computing-components/processors/google-s-tensor-processing-unit-explained-\this-is-what-the-future-of-computing-looks-\like-1326915`.

[2] `https://www.sdxcentral.com/articles/news/intels-deep-learning-chips-will-arrive-2017/2016/11/`.

[3] Suyoung Bang, Jingcheng Wang, Ziyun Li, Cao Gao, Yejoong Kim, Qing Dong, Yen-Po Chen, Laura Fick, Xun Sun, Ron Dreslinski, et al. 14.7 a 288μw programmable deep-learning processor with 270kb on-chip weight storage using non-uniform memory hierarchy for mobile intelligence. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 250–251. IEEE, 2017.

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

[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Notices*, 49:269–284, 2014.

[6] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[7] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceed-*

ings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pages 609–622. IEEE Computer Society, 2014.

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

[9] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

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

[11] Giuseppe Desoli, Nitin Chawla, Thomas Boesch, Surinder-pal Singh, Elio Guidetti, Fabio De Ambroggi, Tommaso Majo, Paolo Zambotti, Manuj Ayodhyawasi, Harvinder Singh, et al. 14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 238–239. IEEE, 2017.

[12] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[13] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pages 92–104. IEEE, 2015.

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

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

[16] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. *CoRR*, abs/1812.01187, 2018.

[17] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

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

[19] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[20] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[21] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. Stripes: Bit-serial deep neural network computing. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12. IEEE Computer Society, 2016.

[22] Cong Leng, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with admm. *arXiv preprint arXiv:1707.09870*, 2017.

[23] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[24] Sijia Liu, Jie Chen, Pin-Yu Chen, and Alfred Hero. Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications. In *International Conference on Artificial Intelligence and Statistics*, pages 288–297, 2018.

[25] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

[26] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kyung Kim, and Hadi Esmaeilzadeh. Tabla: A unified template-based framework for ac-

celerating statistical machine learning. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pages 14–26. IEEE, 2016.

[27] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 246–247. IEEE, 2017.

[28] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7197–7205, 2017.

[29] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[30] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 267–278. IEEE, 2016.

[31] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction method of multipliers. *arXiv preprint arXiv:1812.11677*, 2018.

[32] Mark B. Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[33] Jaehyeong Sim, Jun-Seok Park, Minhye Kim, Dongmyung Bae, Yeongjae Choi, and Lee-Sup Kim. 14.6 a 1.42 tops/w deep convolutional neural network recognition processor for intelligent ioe systems. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, pages 264–265. IEEE, 2016.

[34] Mingcong Song, Kan Zhong, Jiaqi Zhang, Yang Hu, Duo Liu, Weigong Zhang, Jing Wang, and Tao Li. In-situ ai: Towards autonomous and incremental deep learning for iot systems. In

High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on, pages 92–103. IEEE, 2018.

[35] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, et al. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 13–26. IEEE, 2017.

[36] Sebastian Vogel, Jannik Springer, Andre Guntoro, and Gerd Ascheid. Efficient acceleration of cnns for semantic segmentation on fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 309–309. ACM, 2019.

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

[38] Paul N Whatmough, Sae Kyu Lee, Hyunkwang Lee, Saketh Rama, David Brooks, and Gu-Yeon Wei. 14.3 a 28nm soc with a 1.2 ghz 568nj/prediction sparse deep-neural-network engine with¿ 0.1 timing error rate tolerance for iot applications. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 242–243. IEEE, 2017.

[39] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[40] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint arXiv:1611.05128*, 2016.

[41] Chen Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 12. ACM, 2016.

[42] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. Energy-efficient cnn implementation on a deeply pipelined fpga cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 326–331. ACM, 2016.

[43] Hongyi Zhang, Moustapha Cissé, Yann Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2018.

[44] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers, 2018.

[45] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *International Conference on Learning Representations (ICLR)*, 2017.