Mayo: A Framework for Auto-generating Hardware Friendly Deep Neural Networks

Yiren Zhao*
University of Cambridge
Cambridge, UK
yaz21@cam.ac.uk

Xitong Gao†
Shenzhen Institute of Advanced Technology,
Chinese Academy of Sciences
Shenzhen, China
xt.gao@siat.ac.cn

Robert Mullins
University of Cambridge
Cambridge, UK
robert.mullins@cam.ac.uk

Chengzhong Xu
Shenzhen Institute of Advanced Technology,
Chinese Academy of Sciences
Shenzhen, China
cz.xu@siat.ac.cn

ABSTRACT

Deep Neural Networks (DNNs) have proved to be a convenient and powerful tool for a wide range of problems. However, the extensive computational and memory resource requirements hinder the adoption of DNNs in resource-constrained scenarios. Existing compression methods have been shown to significantly reduce the computation and memory requirements of many popular DNNs. These methods, however, remain elusive to non-experts, as they demand extensive manual tuning of hyperparameters. The effects of combining various compression techniques lack exploration because of the large design space. To alleviate these challenges, this paper proposes an automated framework, Mayo, which is built on top of TensorFlow and can compress DNNs with minimal human intervention. First, we present overriders which are recursively-compositional and can be configured to effectively compress individual components (*e.g.* weights, biases, layer computations and gradients) in a DNN. Second, we introduce novel heuristics and a global search algorithm to efficiently optimize hyperparameters. We demonstrate that without any manual tuning, Mayo generates a sparse ResNet-18 that is 5.13× smaller than the baseline with no loss in test accuracy. By composing multiple overriders, our tool produces a sparse 6-bit CIFAR-10 classifier with only 0.16% top-1 accuracy loss and a 34× compression rate. Mayo and all compressed models are publicly available. To our knowledge, Mayo is the first framework that supports overlapping multiple compression techniques and automatically optimizes hyperparameters in them.

CCS CONCEPTS

•Computing methodologies → Computer vision; •Mathematics of computing → *Arbitrary-precision arithmetic;* **•Software and its engineering** → *Software libraries and repositories;*

KEYWORDS

Deep Neural Network, Pruning, Quantization, Automated Hyperparameter Optimization

*† Xitong Gao and Yiren Zhao contributed equally to this work.

1 INTRODUCTION

Deep Neural Networks (DNNs) achieve state-of-the-art performance on both vision [17] and speech [9] applications by leveraging parameter-rich networks learning from massive amount of data. Traditionally, the substantial computational resources required by DNNs are prohibitively expensive in low-powered environments, making the deployment of large neural networks a challenging endeavour.

DNNs are, in general, inherently *redundant*, which means that many operations in DNNs compute highly-correlated results. We can exploit this fact to accelerate DNNs without a detrimental impact on accuracies. Researchers have proposed various compression techniques. For instance, sparsity-inducing regularization methods [21] encourage sparsity in DNNs, enabling us to prune computations by skipping those of negligible importance. Quantization techniques [12] use low-precision arithmetic instead of 32-bit floating-points traditionally used by CPUs and GPUs. Low-rank approximation (LRA) [15] identifies redundancies in DNNs by singular value decomposition, then eliminates them by reducing the rank of the singular matrix. These optimizations directly engender computation or memory savings in a custom hardware design.

Although these techniques demonstrate high effectiveness in compressing DNNs with minimal loss of accuracies, they are often associated with large hyperparameter design spaces. For instance, the compression of different components (weights, activations, *etc.*) in each layer can be configured differently. The search process of suitable hyperparameters that provide high compression rates and minimal impact on accuracies in DNNs is often time-consuming, and demands expertise in the respective methods and the underlying network structure. In addition, it has been shown that many of these compression methods can be used jointly [7]. The compositional nature encourages us to chain them to achieve even higher compression rates, yet it further exacerbates the difficulty in finding the optimal settings, due to the combinatorial explosion of the hyperparameter design space and the fact that the interplay among these compression methods is not well-understood.

In this paper we present a new framework, Mayo, to assist the explorations of various compression techniques with hardware

design in focus, as the first step toward a fully automated DNN architecture optimizer that rivals manual tuning by human experts.

A typical layer computation in a DNN forward pass can generally be represented by $\mathbf{x}_i = f_i(\mathbf{x}_{i-1}, \mathbf{\Theta}_i)$, where of the i-th layer, f_i is the underlying algorithm, $\mathbf{x}_{i-1}$ is the input, and $\mathbf{\Theta}_i$ represents the trainable parameters. Our framework contains a wide range of operations which we call *overriders* that can be fully customized to override various components in the computation above. For instance, the layer computation can be replaced with $\mathbf{x}_i = \tilde{f}_i(\mathbf{x}_{i-1}, \tilde{\mathbf{\Theta}}_i)$, where the parameters $\tilde{\mathbf{\Theta}}_i$ and the function $\tilde{f}_i$ are hardware-efficient variants of $\mathbf{\Theta}_i$ and f_i generated by the respective overriders. Overriders can also be composed by chaining multiple overriders in sequence. Finally, Mayo can efficiently explore the design space of hyperparameters within overriders to achieve high compression rates while minimizing the impact on the accuracies of DNNs. Mayo can be easily applied in the scenarios including but not limited to:

- trade off inference speed and accuracy; and
- slim large models on a smaller dataset with fewer labels using transfer learning [26].

This paper makes the following contributions:

- We introduce the Mayo framework by discussing the supported compression techniques, and explain how we use novel heuristics to automatically derive some hyperparameters based on input conditions (Section 3.1).
- We propose a resource-aware search algorithm that can automatically optimize hyperparameters in overriders while minimizing the impact on accuracies (Section 3.2).
- Building upon the contributions above, we present two case studies. First, we demonstrate how Mayo can obtain state-of-the-art compression rates automatically with fine-grained pruning (Section 6.1). In the second case study, we show the optimization results of chaining pruning and quantization methods, and further demonstrate that non-linear quantization works better than the linear variants on pruned DNNs (Section 6.2).

2 RELATED WORKS

2.1 Compression Techniques

A wide range of compression techniques have proven to be effective for lowering the computation and memory requirements in a pretrained DNN.

Pruning directly reduces the number of connections. Guo *et al.* [5] propose dynamic network surgery for fine-grained pruning, using adjustable threshold conditions to remove individual weights and their connections from the DNN. Mao *et al.* [24] show the extra overhead of fine-grained pruning on SIMD architectures, but suggest that a coarse-grained variant has a faster inference time at the cost of a lower compression rate. The granularity of pruning and the pruning ratio are therefore often varied to provide a suitable trade-off between performance and DNN's accuracy.

Quantization methods enable each parameter to be represented with much narrower bit-width than the 32-bit single-precision floating-point typically used in CPU- or GPU-based DNN implementations. In the extreme case, the parameters can be binary values [11]. This significantly reduces the memory requirements for parameters. Furthermore, quantized intermediate computations in DNNs use low-precision arithmetic, which in turn save computational resources. These methods use number representations and arithmetics based on, for instance, fixed-point arithmetic [6], logarithmic [20], powers of 2 [27], *etc.* Similar to pruning, quantization provides diverse design trade-offs among speed, energy expenditure and accuracy.

Besides these techniques, there are many alternative compression methods. For instance, low-rank approximation [15, 23] reduces the rank of the weight matrix while minimizing the deviation from the original.

Han *et al.* [7] further demonstrate that combining a series of compression algorithms gives even higher compression rates, and show that many popular large-scale DNNs can be compressed in size dramatically. Their methodology, however, tweaks hyperparameters by hand. With compositionality in mind, we design Mayo to scalably explore the design space, steering the interaction of multiple compression methods to automatically optimize the trade-off relationship between speed and accuracy.

2.2 Existing Tool-chains and Frameworks

Various frameworks have been developed to fine-tune DNNs for resource-constrained scenarios. Gysel *et al.* [6] propose *Ristretto* to compress models mainly using fixed-point quantizations. Milde *et al.* [25] introduce *ADaPTION* which also performs fixed-point quantizations, but further extends them to to support stochastic round-off behaviours. Zhou *et al.* [28] present *DoReFa-Net* with *TensorPack*, and they are the first to propose training with limited-precision gradients.

3 OPTIMIZATIONS

Quantization methods allow us to configure hyperparameters (*e.g.* the bit-widths) used by individual components in a DNN, directly reducing the hardware resource usage while increasing the round-off and overflow errors incurred. The resulting search space of hyperparameters to trade off resources with accuracy may however be infeasible to traverse exhaustively. In this section we explain how to automate and accelerate the search procedure. Our approach is two-fold. In Section 3.1, we first introduce various quantization methods used in Mayo, and the corresponding heuristics to sensibly determine some of the hyperparameters. We then continue to explain in Section 3.2 how all remaining hyperparameters are selected automatically.

3.1 Quantizations and Heuristics

For custom hardwares, the most pervasive quantization method is fixed-point quantization. An n-bit fixed-point number with a binary point position p can represent a real value x with:

$$x = 2^{-\mathsf{p}} \times m_1 m_2 \ldots m_{\mathsf{n}}, \tag{1}$$

where the bits $m_1 m_2 \ldots m_n$ denote a binary integer in 2's-compliment. It is clear from Equation (1) that the range of representable values are bounded by $[-2^{-\mathsf{p}}2^{\mathsf{n}}, 2^{-\mathsf{p}}(2^{\mathsf{n}} - 1)]$, and values outside this range are quantized by saturation, *i.e.* represented by one of the bounds accordingly. Moreover, the choice of p also affects accuracy as p yields round-off error bounded by $2^{-\mathsf{p}}$.

We observe that saturated values often have greater impact on accuracy than the precision p used. We thus iteratively adjust p for each set of parameters such that the mean quantization error, *i.e.* the mean of the sums of round-off and saturation error, *i.e.* $\|x - \tilde{x}\|_1$, the ℓ^1-distance between the data and the quantized variant, is minimized. In contrast to Courbariaux *et al.* [3], our algorithm is a search procedure, which prevents p to oscillate between consecutive iterations.

Mayo additionally implements a *mini-float* quantization, which follows the IEEE-754 floating-point standard [14], but allows arbitrary bit-widths to be used. A mini-float can represent real values with:

$$x = (-1)^s \times 2^{e-\mathrm{b}} \times 1.m_1 m_2 \ldots m_{\mathrm{p}}, \tag{2}$$

where s is the sign bit, the exponent e is a k-bit non-negative integer, the bias b is a constant offset applied to e, and the p-bit $m_1 m_2 \ldots m_{\mathrm{p}}$ represent the mantissa bits, here $1.m_1 m_2 \ldots m_{\mathrm{p}}$ indicates an unsigned fixed-point number in binary.

A mini-float quantization scheme (k, b, p) has three hyperparameters, the combinations of all possible values are thus infeasible to explore exhaustively. We therefore reduce the search space to the bit-width $\mathsf{n} = \mathsf{k} + \mathsf{p}$, and algorithmically determine the quantization scheme from n and the input data by minimizing the incurred quantization error. Our approach is similar to the algorithm described for fixed-point. For a given bit-width n, we iterate through mantissa widths $\mathsf{p} \in [0, \mathsf{n}]$, and use $\mathsf{k} = \mathsf{n} - \mathsf{p}$ as the exponent widths. With given k and p, it is now possible to compute a bias b so that no overflow occurs in the quantized values. Using the above (k, b, p), we can deduce the round-off error in a way similar to the fixed-point case, and finally find the hyperparameter combination that minimizes the error.

Quantization with powers of 2 is a degenerate form of the mini-float that uses no mantissa, *i.e.* $\mathsf{p} = 0$. Our heuristic search process also works for this specialized form to search for b, given k and the input data. This method results in the following representable values:

$$x = (-1)^s \times 2^{e-\mathrm{b}}. \tag{3}$$

In hardware implementations, multiplications with powers of 2 can be achieved with barrel shifters, which in general are much cheaper than multipliers.

It is noteworthy that Mayo approximates all quantization methods using floating-point numbers, this approximation may introduce round-off error that are not captured in this tool.

3.2 Automated Hyperparameter Optimization

The techniques detailed above reduce quantization design to the choice of quantization bit-widths, significantly reducing the size of the hyperparameter exploration space. In this section we further extend our methodology to automatically explore the remaining hyperparameters.

Without a loss of generality, here we adopt the view that a hyperparameter γ is associated with a predefined performance penalty cost(γ) monotonic to γ. For instance, we may use the computational or memory utilization estimates as the penalty function of γ. With this notion, we greedily decrement the hyperparameter γ_t with the highest cost until the accuracy requirement can no longer be met.

Algorithm 1 Hyperparameter Optimization

```
1:  procedure OPTIMIZE(N, α_budget, Γ, Δ, s)
2:      B ← ∅
3:      while Γ changed ∨ I(Γ)/B ≠ ∅ do
4:          t ← argmax_{t ∈ I(Γ)/B} cost(γ_t)
5:          γ_t ← γ_t − δ_t
6:          α ← train(N, Γ, s)
7:          if α ≥ α_budget then
8:              continue
9:          if δ_t ≥ ε_t then
10:             backtrack(N, Γ)
11:             δ_t ← δ_t / 2
12:         else
13:             B = B ∪ {t}
```

Algorithm 1 is a high-level description of our greedy search with backtracking, which accepts a DNN N and produces a fine-tuned N with a set of minimized hyperparameters Γ as its output, while satisfying the desirable accuracy target α_{budget}. Here, the input Γ initializes all hyperparameters to their upper bounds, and Δ specifies the stride sizes used to decrement hyperparameters. The function train(N, Γ, s) fine-tunes N with Γ for s steps and returns the final accuracy of N, $\mathcal{I}(\Gamma)$ denotes the indices that can be used to address hyperparameters, backtrack(N, Γ) returns N and Γ to their previous states. Finally, $\delta_t \in \Delta$ is the stride used to decrement γ_t, ϵ_t is the lower bound on δ_t, and B blacklists hyperparameters that can no longer be minimized without degrading the accuracy below α_{budget}.

4 FEATURES

Mayo is built on top of TensorFlow [1] and specializes in DNN compression with quantization methods in Section 3.1, pruning techniques [5], LRA [15], *etc.* These methods are implemented as objects called *overriders* in Mayo. Overriders can be flexibly applied to not only parameters $\mathbf{\Theta}_i$, but also the underlying algorithm f_i, and even the gradient of each layer computation $\mathbf{x}_i = f_i(\mathbf{x}_{i-1}, \mathbf{\Theta}_i)$. The design of overriders provides an abstraction for various compression techniques.

An overrider $g \in G$, configured with suitable hyperparameters, takes a multi-dimensional array as input, and produces a new array with the same shape as the compressed variant. Parameters $\mathbf{\Theta}_i$ can thus be simply substituted using any overrider $g_{\text{param}} \in G$:

$$\tilde{\mathbf{\Theta}}_i = g_{\text{param}}(\mathbf{\Theta}_i). \tag{4}$$

Moreover, overriders can be used to customize other components in a DNN. For example, we can customize the activation function to replace f_i with $\tilde{f}_i$, using an overrider $g_{\text{activation}} \in G$, where for any input $\mathbf{x}_{i-1}$ and parameters $\mathbf{\Theta}_i$, we have:

$$\tilde{f}_i(\mathbf{x}_{i-1}, \mathbf{\Theta}_i) = g_{\text{activation}}\left(f_i\left(\mathbf{x}_{i-1}, \mathbf{\Theta}_i\right)\right). \tag{5}$$

Finally, overriders are recursively-compositional. Multiple overriders can be chained in sequence, which in turn provides greater compression opportunities. For example, given a pruning overrider g and a quantizing overrider h, the composition of them is also an overrider that can be applied to any components, *i.e.* $h \circ g \in G$. In

		Mayo	Ristretto	ADaPTION	DoReFa
Pruning	fine-grained	✓	✗	✗	✗
	coarse-grained	✓	✗	✗	✗
Quantization	fixed-point	✓	✓	✓	✓
	dynamic fixed-point	✓	✓	✓	✗
	mini-float	✓	✓	✓	✗
	log	✓	✗	✗	✗
	shift	✓	✓	✓	✗
Layer-wise customization		✓	✓	✓	✗
Automated hyperparameter optimization		✓	✗	✗	✗
Compression method chaining		✓	✗	✗	✗
Customizable components		w/a/g	w/a	w/a	w/a/g
Configuration format		YAML	Caffe	Caffe	Python

Table 1: A comparison: Ristretto [6], ADaPTION [25], DoReFa [28] and Mayo.

this case, for any parameters Θ:

$$\tilde{\Theta} = h\left(g\left(\Theta\right)\right). \tag{6}$$

In Mayo, DNNs are described in a readable format called *YAML* [2]. Individual components (*e.g.* weights, biases, activations and gradients) of each layer can be flexibly customized by specifying the overriders and the associated hyperparameters to use. Each component in each layer can therefore be customized differently by having different overriders.

5 COMPARISONS TO EXISTING FRAMEWORKS

Table 1 compares the features of Mayo to the compression frameworks mentioned in Section 2.2. Here, the customizable components w, a and g respectively denote the weight parameters, activation and gradients of each layer computation. Most of the existing frameworks focus purely on quantizations. In contrast, quantization is just one of the many classes of overriders in Mayo. Mayo further supports other compression techniques such as fine- and coarse-grained pruning, and LRA. Additionally, Mayo is highly flexible: it can customize the compression techniques used by any individual components in each layer. It is also the first tool that supports chaining multiple compression techniques.

Mayo further automates hyperparameter optimization. Both *Ristretto* and *ADaPTION* support fine-tuning quantized DNNs, but the process of manual hyperparameter optimization is often time-consuming and requires extensive knowledge in both the compression method and the underlying network structure. For example, a common flow in *ADaPTION* is to manually allocate bit-widths in a layer-wise manner and tweak them repeatedly until the accuracy and bit-width criteria are met. Mayo completely automates the manual process, and provides a trade-off between the compression rate and test accuracy.

6 EXPERIMENTS

In this section, we present two case studies. The first one applies fine-grained pruning on a wide range of vision DNNs to demonstrate our automated hyperparameter optimization in Section 3.2. The second showcases the quantization methods in Section 3.1, and examines the effects they have on a DNN and its pruned variant.

6.1 Fine-grained Pruning

In this case study, we override weight parameters of each layer using a fine-grained pruning method known as dynamic network surgery (DNS) [5], where each layer is associated with a hyperparameter to trade-off sparsity with the network accuracy. We use automated hyperparameter optimization to compress a wide range of DNNs. Our optimization is resource-aware, as we define the penalty function of each hyperparameter to be the number of remaining active parameters in the layer. Table 2 shows 2–90× compression rates on them, where ER and CR respectively denote error rate and compression rate. Here, LeNet-5 [18] classifies the MNIST dataset [19]. CifarNet is a custom-built classifier for the CIFAR-10 dataset [16]. Finally, AlexNet [17], SqueezeNets [13] 1.0 and 1.1, MobileNetV1 [10], and ResNet-18 [8] classify images in ImageNet [4]. Moreover, Mayo fully supports the depthwise-separable convolution layers and residual connections respectively found in MobileNetV1 and ResNet-18.

Table 3 shows our automated pruning comparing against other published results with manual optimization. The first (Naïve) and the second (Deep Compression) methods produced by Han *et al.* [7] respectively show fine-grained pruning with and without fine-tuning to regain the lost accuracy due to pruning. As Han *et al.* applies multiple compression passes, we only consider their pruning results for a consistent comparison. The third method is the original DNS used by Guo *et al.* [5]. They manually tailored the hyperparameters to obtain a compression rate of 17.7×. In Mayo, although the top-1 error increases by 1.05%, we managed a much higher compression rate at 21.8×, as we allow each layer to adopt different hyperparameter values. For the larger DNNs such as MobileNetV1

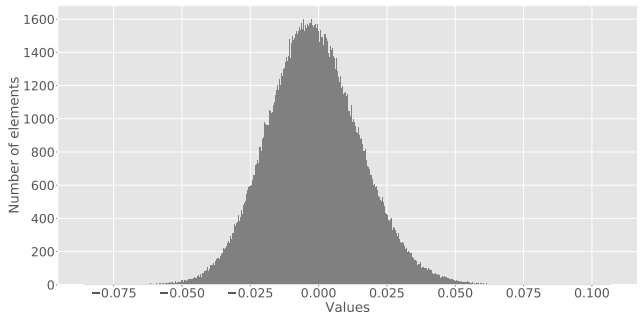
and ResNet-18, Mayo respectively achieves compression rates of 2.97× and 5.13×, with top-1 accuracy losses of only 0.76% and −0.28%.

Model Name	Original ER	Increase in ER	CR
LeNet-5 [18]	0.7%/0%	0%/0%	90.1×
CifarNet	8.63%/0.34%	0.25%/-0.04%	6.39×
AlexNet [17]	44.14%/21.40%	1.27%/0.20%	21.83×
SqueezeNet 1.0 [13]	43.55%/20.76%	0.27%/0.11%	2.07×
SqueezeNet 1.1 [13]	43.01%/20.22%	0.73%/0.36%	2.16×
MobileNetV1 [10]	29.25%/10.47%	0.76%/1.58%	2.97×
ResNet-18 [8]	31.02%/11.32%	-0.28%/-0.55%	5.13×

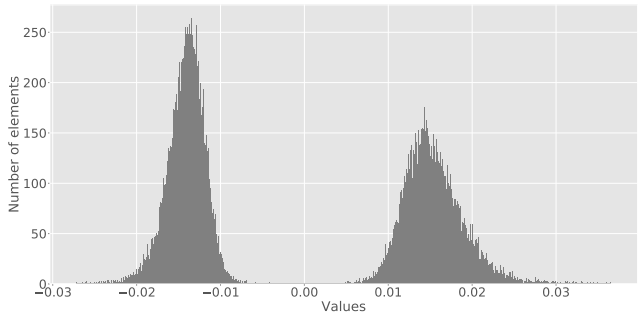
Table 2: Fine-grained pruning with automated hyperparameter optimization.

Model	Error rates	Compression rate
Naïve [7]	57.18%/23.23%	4.4×
Deep Compression [7]	42.77%/19.67%	9×
DNS (Original) [5]	43.09%/19.99%	17.7×
DNS (Mayo)	44.14%/21.40%	21.8×

Table 3: A comparison of fine-grained pruning results on AlexNet [17].



(a) Dense weights



(b) Sparse weights

Figure 1: Weight distributions of the final convolutional layer in dense and sparse CifarNets.

6.2 Quantized Sparse and Dense Models

Our next case study is based on CifarNet, which has a top-1 accuracy of 91.37% and only 1.3M parameters. Our baseline is significantly smaller than the VGG-based CIFAR-10 classifier (top-1 93.66% with 20M parameters) in [22]. We consider a pretrained CifarNet and its pruned variant as the dense and sparse models, and respectively apply fixed-point, dynamic fixed-point, shift and mini-float quantizations.

The quantization results on the sparse and dense CifarNet baselines are shown in Table 4. For the dense model, Mayo quantizes the networks to a bit-width of 4, and the sparse variants are quantized to 6.

Figure 1 shows the weight distributions of the pre-quantized baselines. For the dense model, a large number of parameters are centralized around zero and quantization methods that can faithfully represent near-zero values give better test accuracies (mini-float and shift). In contrast, representable fixed-point values distribute evenly across the entire quantized range, and thus they struggle to accommodate a largely uneven distribution. Shift quantization shows the worst performance on sparse models but has relatively good performance on dense models. As it quantizes values close to zero with much less round-off error, it fits the weights distribution of the dense models better than that of the sparse variant. Overall, for CifarNet, Mayo achieves a compression rate of 33.92x with only 0.16% top-1 accuracy loss by jointly using fine-grained pruning and mini-float quantization.

7 CONCLUSION

In this paper, we present the Mayo framework that can automatically compress a pretrained neural network for saving the computational and memory resources. It optimizes the quantization procedure and reduces the exploration space. In addition, it performs a greedy search to find optimal combinations of parameters for multiple overriders, thus fully automates the process of model compression. In terms of features, Mayo specializes in chaining various compression techniques and can flexibly customize individual components (weights, activations, gradients, *etc.*) using them in a DNN.

The results demonstrate that Mayo can achieve compression rates exceeding the previous state-of-the-art results using fine-grained pruning, without any manual hyperparameter tuning. Furthermore, we show results of different quantization methods on an original and pruned DNN, and observe that non-linear approaches provides better trade-offs between accuracy and compression rate than linear quantization for both sparse and dense models. Mayo and the compressed networks used in the paper are released to the public.*

*Available at: https://github.com/deep-fry/mayo.

Method	Bit-width	Density	Compression rate	Top-1/top-5 accuracies
Baseline	32	100%	-	91.37%/99.67%
Fixed-point (Fixed-p)	4	100%	8×	89.64%/99.74%
Dynamic Fixed-point (DFP)	4	100%	8×	90.63%/99.68%
Shift	4	100%	8×	91.16%/99.65%
Mini-float (MF)	4	100%	8×	91.83%/99.72%
Fine-grained pruning (FPrune)	32	15.65%	6.39×	91.12%/99.70%
FPrune + Fixed-p	6	15.65%	33.92×	90.59%/99.68%
FPrune + DFP	6	15.65%	33.92×	91.04%/99.70%
FPrune + Shift	6	15.65%	33.92×	89.28%/99.68%
FPrune + MF	6	15.65%	33.92×	91.21%/99.73%

Table 4: Quantizations on the sparse and dense CifarNets.

ACKNOWLEDGMENTS

This work is supported by the Chinese National Basic Research Program (973 Program, No. 2015CB352400), the National Natural Science Foundation of China (Grant U1401258), and the Science and Technology Planning Project of Guangdong Province (2015B010129011). We thank EPSRC for providing Yiren Zhao his doctoral scholarship.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang. TensorFlow: A system for large-scale machine learning. In *USENIX Conference on Operating Systems Design and Implementation*, 2016.

[2] O. Ben-Kiki, C. Evans, and B. Ingerson. YAML ain't markup language. *yaml.org*, 2005.

[3] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. In *International Conference on Learning Representations*, 2015.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[5] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient DNNs. In *Advances in Neural Information Processing Systems*, 2016.

[6] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[7] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[9] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*. 2016.

[12] K. Hwang and W. Sung. Fixed-point feedforward deep neural network design using weights +1, 0, and −1. In *Signal Processing Systems*, 2014.

[13] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[14] IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, 2008.

[15] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *CoRR*, abs/1405.3866, 2014.

[16] A. Krizhevsky, V. Nair, and G. Hinton. The CIFAR-10 and CIFAR-100 datasets. *http://www.cs.toronto.edu/ kriz/cifar.html*, 2014.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 2012.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits.

[20] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong. LogNet: Energy-efficient neural networks using logarithmic computation. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017.

[21] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy. Sparse convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[22] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *International Conference on Computer Vision*, 2017.

[23] P. Maji and R. Mullins. 1D-FALCON: Accelerating deep convolutional neural network inference by co-optimization of models and underlying arithmetic implementation. In *International Conference on Artificial Neural Networks*, 2017.

[24] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally. Exploring the regularity of sparse structure in convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[25] M. B. Milde, D. Neil, A. Aimar, T. Delbrück, and G. Indiveri. ADaPTION: Toolbox and benchmark for training convolutional neural networks with reduced numerical precision weights and activation. *CoRR*, abs/1711.04713, 2017.

[26] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 2010.

[27] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless CNNs with low-precision weights. *International Conference on Learning Representations (ICLR)*, 2017.

[28] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.