Pruning and Quantization Enhanced Knowledge Distillation

Wenjie Wang, Fengbin Zhu, Yujie Zhang, Yichen Zhou
National University of Singapore

Abstract

Deep compression is important for modern deep learning models in various industrial applications. There are several different types of deep compression techniques that aim to reduce the size of the network and accelerate the model inference, such as Knowledge Distillation (KD), Pruning, and Quantization. However, existing works seldom study the combination of multiple different techniques. In this work, we aim to explore and verify the effectiveness of combining different KD methods with various compression techniques including pruning, weight sharing and quantization. We conducted comprehensive experiments for several compression pipelines with two or three compression steps on CIFAR10 [7] with ResNets [4]. We demonstrate that pruning and quantization enhanced KD can further compress the student model while maintaining the performance. Besides, KD methods perform differently when incorporating various compression techniques. The insights shed light on how to effectively incorporate various deep compression techniques when training deep learning models.

1 Introduction

Great progress has been made in deep learning to solve many different tasks such as computer vision, speech and natural language processing. While the state-of-the-art performance has been constantly improving over the years, the problem of deploying large deep learning models to production system has been a obstacle that limits the usability for real world applications. For many cases, the model inference has to be performed locally, sometimes on mobile or even edge devices with limited memory and/or computing power. Therefore, we need to reduce the size of deep models in order to make them fit into the device's memory and ensure low latency and high throughput required by the use case.

Several deep compression techniques were proposed by Han *et al.* [2] for reducing the size of a large trained deep learning model. They proposed a three stage pipeline including pruning, trained quantization and Huffman coding, which can effectively reduce the model size by around 40 times without reducing accuracy. Since then, several works including but not limited to [3, 13, 10, 15, 11] explored different ways of pruning the filters to reduce the model size and improve the inference speed. Moreover, there are also several developments in quantization techniques which reduce the number of bits per weight from 32 to 3 [9] or even 2 [6]. Another major family of model compression method is Knowledge Distillation (KD) [5], which tries to distill the knowledge learned by a large teacher model into a smaller student model. Several works such as [12, 8] recently propose more sophisticated methods to improve the knowledge distillation process in an effort to further reduce the model size while maintaining the accuracy of the original teacher model.

To our best knowledge, most existing works only try to improve the effectiveness of one particular type of compression method, *e.g.,* only KD. It is questionable whether such improvements in one particular method can still work well when combined with other compression algorithms which are usually applied to the raw model. Therefore, in this work, we would like to explore and verify the effectiveness of combining different KD methods with various compression techniques including

Preprint. Under review.

pruning, weight sharing and quantization. We build several model compression pipelines by using two or three different model compression techniques, and performed comprehensive experiments on CIFAR10 [7] with ResNets [4]. The results indicate that pruning and quantization enhanced KD can improve the compression ratio of the student model while keeping the performance at the same level of the original model. Moreover, we notice that some KD methods do not seem to perform well when applied with other compression methods. We summarize the observations from our experiments and hope that these insights can be helpful for researchers and engineers in building effective deep learning model compression pipelines.

2 Related Work

2.1 Pruning

As a direct and simple method to compress neural networks, pruning has been widely studied by researchers. By pruning the unimportant connections, a non-structured sparsity method is first proposed in [3] to reduce the model storage and computation by an order of magnitude with no loss of accuracy. Besides, Wen *et al.* [13] proposes a Structured Sparsity Learning (SSL) method to get a compact and hardware-friendly structure of CNNs by adaptively regularizing the filters, channels, filter shapes and layer depth. SSL method not only speeds up the model computation, but also improves the classification accuracy. Additionally, Li *et al.* [10] prunes whole filters in CNN with a little bit loss of the model accuracy but it achieves a better acceleration in GPUs and CPUs. Zhu *et al.* [15] develops a gradual pruning approach with minimal tuning. Liu *et al.* [11] enforces channel-level sparsity in the network by automatically identifying insignificant channels and then pruning them. All those pruning methods can reduce the model memory footprint while maintaining the original accuracy or with a little accuracy degradation.

2.2 Quantization

Quantization is able to compress neural networks by reducing the number of bits for parameter representation. Hubara *et al.* [6] uses the binary weights and activations to compute the parameter gradients during training. By replacing the most arithmetic operations with bit-wise operations, it significantly simplifies the computation. Li *et al.* [9] proposes ternary weight networks with weights constrained to +1, 0 or -1. By minimizing the Euclidian distance between ternary weights and full precision weights, and optimizing a threshold-based ternary function, the effective approximation of full precision weights can be computed fast and easily. However, those quantization methods unavoidably cause accuracy degradation to some extent. To alleviate this issue, Dong *et al.* [1] introduces a stochastic quantization algorithm to quantize a part of elements according to the quantization error randomly and iteratively, which improves the model accuracy while maintaining the low storage and high computing efficiency. Besides, Zhang *et al.* [14] designs a co-training process for a quantized DNN and its corresponding quantizers, discarding the original fixed and handcrafted quantization schemes such as uniform or logarithmic quantization.

2.3 Knowledge Distillation

KD aims to compress deep models by extracting the valuable information from teacher model into a compact student model. The target is to get a smaller model but with comparable accuracy with teacher model. The classic work [5] is proposed by Hinton *et al.* , who train the student model using a weighted average of two different objective functions: the cross entropy loss functions with the soft predictions from the teacher model and the correct labels as the target, respectively. Romero *et al.* [12] adopts two-stage model training by firstly pre-training the student network to learn an intermediate representation of the teacher network, and then performing a KD training of the whole network. Recently, stage-wise KD method has been proposed in [8]. By stage-wise KD training of feature maps and independent training of classifier, the trained student model gets superior accuracy with less data.

3 Method

3.1 Overall Framework

In this paper, we construct a three-stage pipeline to enhance KD with pruning and quantization for further compression of the trained student model. As illustrated in Figure 1, we firstly utilize KD to learn the valuable supervision information from the pre-trained teacher model. After KD, we select the best student model in the first stage. And then several compression techniques are applied to this student model to further reduce model size, such as pruning and quantization. Finally, the compressed student model is retrained on the training data for several epochs to improve performance.

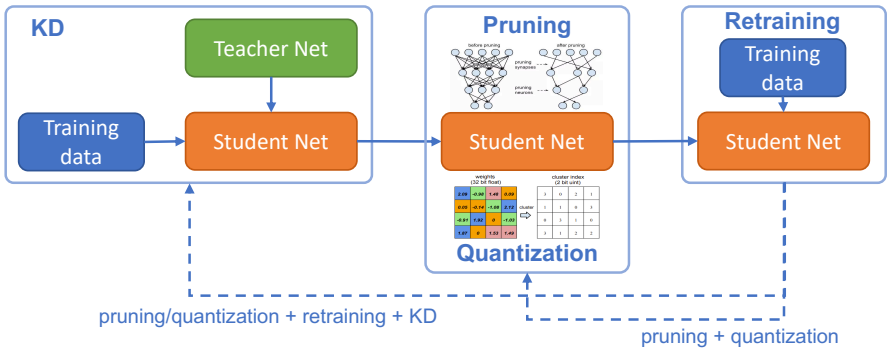

Figure 1: The proposed three-stage pipeline. The student network is firstly trained by KD, and then further compressed by pruning or quantization. Finally, the student network is retrained on the training data.

To further explore the advantages of compositional model compression methods, we enhance KD by pruning and quantization. In the first stage, KD training, we incorporate two popular algorithms, HintonKD [5] and FitNets [12], to transfer knowledge from the pre-trained teacher model. Next, we consider three compression methods: pruning [3], weight sharing [2] and quantization [6]. In the retraining stage, we utilize *Quantization Aware Training (QAT)* for quantization-enhanced KD. For others, we normally retrain the compressed model with the training data.

3.2 KD

HintonKD [5] and FitNets [12] are two kinds of important knowledge distillation methods. HintonKD [5] tries to generate soft targets similar to teacher model, thus extracting more detailed parameters information into student model. As to FitNets[12], in addition to producing similar soft targets, it also learns the intermediate presentations from the teacher to further optimize the student model. Next, we detail the key loss functions for easier comparison.

HintonKD For HintonKD [5], in order to transfer the generalization ability from the teacher to the student, a weighted average of two different loss functions is used for model training. Formally,

$$\mathcal{L}_{KD}(W_S) = \mathcal{H}(y_{true}, P_S) + \lambda \mathcal{H}(P_T^\tau, P_S^\tau), \tag{1}$$

where W_S is the parameters of the student model, $\mathcal{H}$ denotes the cross-entropy loss and λ refers to the hyper-parameter to balance two functions. As P_S is the output probability by the student model, the first loss function forces the student model to predict the correct labels. Besides, P_T^τ and P_S^τ refer to the output probabilities from the teacher model and student model, respectively, which are computed by the same temperature in the softmax. Compared to the hard targets, soft targets help student learn more sophisticated information from teacher model, such as how to identify and process the feature data effectively. Accordingly, the model training efficiency and accuracy can be greatly improved.

FitNets There is a two-stage training in FitNets, including a hint-based training and a KD training of the whole network. The second KD training stage is the same to HintonKD (Equation 1). For the hint-based training, the student model is pretrained up to the guided layers based on the prediction

error of the teacher's hint layers, according to the loss function in Equation 2.

$$\mathcal{L}_{HT}(W_{Guided}, W_r) = \frac{1}{2} \left\| \mu_h(x; W_{Hint}) - r(v_g(x; W_{Guided}); W_r) \right\|^2, \tag{2}$$

where μ_h and v_g denote the teacher/student deep nested functions [12] up to their respective hint/guided layers with parameters W_{Hint} and W_{Guided}, r is the regressor function on top of the guided layer with parameters W_r[12]. This loss function guarantees that student and teacher model can have similar intermediate output for the same training case. In contrast to teaching by the final output probability of last layer, this method can extract valuable information from teacher model quickly and accurately.

3.3 Pruning

Pruning is widely used to sparsify neural networks by reducing the less important parameters so that deep models can be deployed on small mobile devices with less memory and computation consumption. In this work, we prune the parameters with the lowest L1-norm to further compress the neural networks after KD. We set a pruning ratio to control the proportion of pruned parameters in the student model. After pruning, we retrain the pruned model by the supervised training with training data. Note that the pruned parameters will not be used in the retraining stage.

3.4 Weight Sharing

Weight sharing technique is proposed in [2] by classifying weights of each layer into several groups and using the centroids of one group to represent all weights in this group. So, we just need to store the weight indexes for connections and the less number of shared weights, which greatly decreases the model storage requirement. Generally, the model performance after weight sharing is not optimal. A retraining process is still needed to retune the value of all centroids to optimize the neural network. During the forward phase in retraining process, we use the connection index to look up the weight table, getting the corresponding centroid for later computation. While for back-propagation phase, the gradients for all connections in one group are calculated and summed to update their corresponding centroid. The detailed technique implementation is the same as that in [2].

3.5 Quantization

Quantization aims to reduce the model size and accelerate the efficiency of model inference by representing the model parameters at a lower bitwidth than default 32-bit floating-point precision. Existing quantization techniques roughly have two approaches *i.e., Post-training Quantization* and *Quantization Aware Training*. *Post-training Quantization* quantizes the weights and activations of a trained model and QAT maintains the effects of quantization during model retraining. Comparably, the former is easier to use while the latter often enjoys higher accuracy. In this work, we apply both quantization techniques on knowledge distilled models, tending to further improve the model efficiency while maintaining the accuracy.

3.6 Further Compression

We have discussed the compression pipelines involving two model compression methods. Besides, we would like to further explore the compatibility of combining more than two compression algorithms. In this work, we propose to assess the performance of the following two three-stage pipelines:

1. KD + QAT + Pruning;
2. KD + Pruning + Weight Sharing.

The respective algorithms used in these pipelines are the same as those described in previous subsections, we mainly add one extra stage to compress the model further.

4 Experiment

In this section, we introduce our experiments that combine KD with various deep compression techniques, including pruning, weight sharing and quantization.

Settings. We demonstrate the effectiveness of the proposed three-stage pipeline on the popular image classification task. The testing neural network is ResNet [4]. Following prior work [8], we choose ResNet32 as the teacher model and use ResNet10, ResNet18, and ResNet26 as the student model. The image classification benchmark CIFAR10 [7] is applied to evaluate the performance. Since it is a relatively balanced dataset, we only introduce accuracy (Acc) as the evaluation metric. In addition, we incorporate two kinds of baselines: 1) No-teacher. The student model is purely trained from the training data without the help from the teacher model. 2) The student model is only trained by the KD methods, including HintonKD and FitNets.

4.1 KD with Pruning

The pruning ratio is applied to control the proportion of pruned parameters. Figure 2 and 3 visualize the performance comparison when the pruning ratio is set as 0.3 and 0.5, respectively. From the two figures, we can have the following observations: 1) pruning will degrade the model performance, which is reasonable because many parameters are removed. 2) FitNets improves the performance of the student model over No-teacher while HintonKD hurts the accuracy. This might because FitNets has the supervision of feature maps from the teacher, which transfers key knowledge to the student model. 3) More importantly, the performance after retraining is comparable with that of the original model. It is even better over No-teacher and HintonKD, indicating the effectiveness of the proposed three-stage pruning-enhanced KD. Indeed, the pruning can reduce the model complexity and improve the robustness of the student model. Besides, retraining provides the ability for the student model to adjust the parameters.

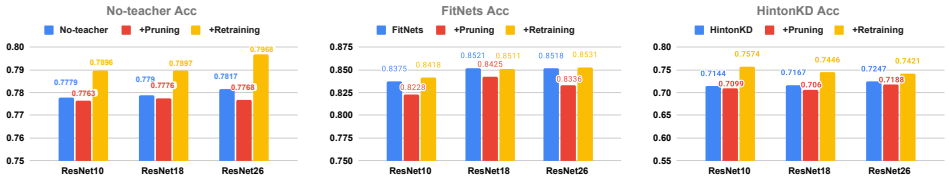

Figure 2: Performance comparison under different KD methods when the pruning ratio is set as 0.3.

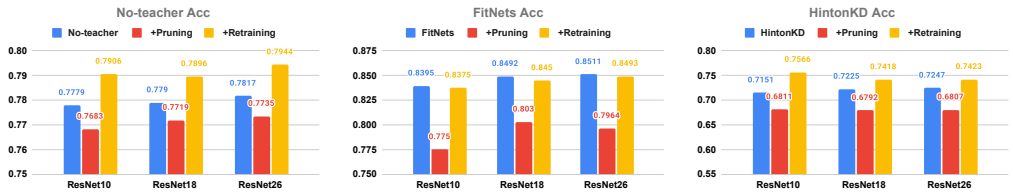

Figure 3: Performance comparison under different KD methods when the pruning ratio is set as 0.5.

4.2 KD with Weight Sharing

In this part, we conducted four groups of experiments by exhibiting connections in each layer with 4 bits, 5 bits, 6 bits or 8 bits index representation, respectively. In other words, we utilized the k-means clustering to classify weights of each layer into 16, 32, 64 or 256 groups and retrained the original weight representation for layer with less parameters. Figure 4 and Figure 5 illustrate the performance comparison under different KD methods for 4 bits and 6 bits index representation, respectively. Other experimental results for 5 bits and 8 bits index representation are listed in Table 1. For all three models, FitNets always get higher accuracy than other two methods. And after retuning, the accuracy degradation from weight sharing is negligible. Especially for 4 bits index representation, we can achieve 7.99x compression rate with accuracy loss less than 1%. Besides, for all three models, No-teacher can obtain a higher accuracy after retaining no matter how many bits are used for index representation. While in terms of HintonKD, for all three models and all four kinds of bits representations, it performs increasingly worse along with stage-wise training. We suspect that compared with other two methods, the accuracy of HintonKD is improved slower under same number of epochs training. In this case, shared weights of each layer cannot indicate the value of

their corresponding connections and constrained by the parameter update rule based on this weight classification, retraining will further worsen the model performance.

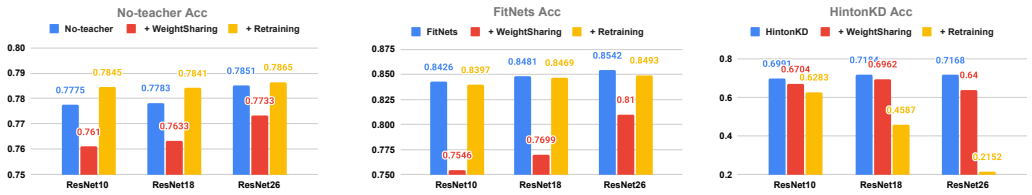

Figure 4: Performance comparison for different KD methods with at most 16 shared weights in each layer (compression rate: 7.99x).

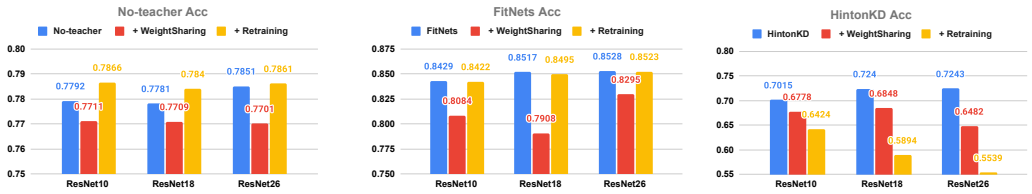

Figure 5: Performance comparison for different KD methods with at most 64 shared weights in each layer (compression rate: 5.32x).

Compression	Model	ResNet10	ResNet18	ResNet26
WS: 5 bits Compression 6.39x	No-teacher	0.7775	0.7783	0.7809
	+ Weight Sharing	0.7729	0.7639	0.7657
	+ Retraining	**0.7835**	**0.784**	**0.7848**
	FitNets	**0.8422**	**0.8506**	**0.8515**
	+ Weight Sharing	0.8078	0.8022	0.8165
	+ Retraining	0.8344	0.8385	0.8421
	HintonKD	0.7014	0.7198	0.7201
	+ Weight Sharing	0.6771	0.6908	0.661
	+ Retraining	0.6114	0.6323	0.2892
WS: 8 bits Compression 3.99x	No-teacher	0.7775	0.7783	0.7809
	+ Weight Sharing	0.7701	0.7692	0.7697
	+ Retraining	**0.7841**	**0.7821**	**0.7848**
	FitNets	**0.8409**	**0.8517**	**0.8517**
	+ Weight Sharing	0.8101	0.8173	0.8125
	+ Retraining	0.8333	0.8421	0.8464
	HintonKD	0.6995	0.7194	0.7191
	+ Weight Sharing	0.6883	0.6897	0.6441
	+ Retraining	0.6709	0.5199	0.3523

Table 1: Model accuracy after combining KD and weight sharing.

4.3 KD with Quantization

In this subsection, we describe our experiments by combining KD and quantization techniques, including Post-training Quantization and Quantization Aware Training. Particularly, we respectively apply Post-training Quantization and QAT on No-teacher model and student models trained using FitNets and HintonKD. From the experimental result, as shown in Table 2, we can observe that applying Post-training Quantization to the No-teacher model and the student model trained using HintonHD results in a slight decrease of accuracy. After employing QAT to further train both models, they can gain significant performance enhancement. Meanwhile, our experiments also show that

applying quantization on FitNets would seriously afflict its performance, especially when the model size is large.

Model	ResNet10		ResNet18		ResNet26	
	Acc	Size (MB)	Acc	Size (MB)	Acc	Size (MB)
No-teacher	0.7835	19.92	0.7814	45.03	0.7803	70.14
+ Quantization	0.7815	4.99	0.7739	11.27	0.7637	17.55
+ QAT	**0.7876**	4.99	**0.7938**	11.27	**0.7943**	17.55
FitNets	**0.8460**	19.92	**0.8490**	45.03	**0.8567**	70.14
+ Quantization	0.8388	4.99	0.2086	11.27	0.5050	17.55
+ QAT	0.8366	4.99	0.6690	11.27	0.5768	17.55
HintonKD	0.7013	19.92	0.7222	45.03	0.7226	70.14
+ Quantization	0.6989	4.99	0.7189	11.27	0.7171	17.55
+ QAT	**0.7906**	4.99	**0.7931**	11.27	**0.7856**	17.55

Table 2: Experimental result by combining KD and quantization.

4.4 Further Compression

4.4.1 KD + QAT + Pruning

Similar to the KD with QAT as mentioned in 4.3, we first perform different knowledge distillation techniques with Quantization Aware Training. After that, we further perform weight pruning with different pruning ratio. The detail results are listed in Table 3. As we can see from the table, for most KD methods and models, the pruning process hardly degrade the performance of a quantized model. When we set a small pruning ratio (0.1) for the QAT trained model, the performance might even increase. The results therefore shows that pruning and quantization can work well together. Hence, it is beneficial to first apply quantization followed by pruning, which can achieve higher compression ratio without sacrificing model performance.

Model	ResNet10		ResNet18		ResNet26	
	Acc	Size (MB)	Acc	Size (MB)	Acc	Size (MB)
No-teacher	0.7835	19.92	0.7814	45.03	0.7803	70.14
+ Quantization	0.7815	4.99	0.7739	11.27	0.7637	17.55
+ QAT	**0.7876**	4.99	0.7938	11.27	**0.7943**	17.55
+ QAT & Pruning (ratio = 0.1)	0.7856	4.49	0.7933	10.14	0.7934	15.80
+ QAT & Pruning (ratio = 0.3)	0.7844	3.49	**0.7941**	7.89	0.7909	12.29
+ QAT & Pruning (ratio = 0.5)	0.7790	2.50	0.7839	5.64	0.7802	8.78
FitNet	**0.8460**	19.92	**0.8490**	45.03	**0.8567**	70.14
+ Quantization	0.8388	4.99	0.2086	11.27	0.5050	17.55
+ QAT	0.8366	4.99	0.6690	11.27	0.5768	17.55
+ QAT & Pruning (ratio = 0.1)	0.8356	4.49	0.6717	10.14	0.5782	15.80
+ QAT & Pruning (ratio = 0.3)	0.8352	3.49	0.6685	7.89	0.5675	12.29
+ QAT & Pruning (ratio = 0.5)	0.8187	2.50	0.6565	5.64	0.5553	8.78
HintonKD	0.7013	19.92	0.7222	45.03	0.7226	70.14
+ Quantization	0.6989	4.99	0.7189	11.27	0.7171	17.55
+ QAT	**0.7906**	4.99	0.7931	11.27	0.7856	17.55
+ QAT & Pruning (ratio = 0.1)	0.7885	4.49	**0.7962**	10.14	**0.7869**	15.80
+ QAT & Pruning (ratio = 0.3)	0.7876	3.49	**0.7962**	7.89	0.7843	12.29
+ QAT & Pruning (ratio = 0.5)	0.7761	2.50	0.7828	5.64	0.7733	8.78

Table 3: Experimental result by applying QAT and pruning techniques.

4.4.2 KD + Pruning + Weight Sharing

We also performed another set of experiments of KD + Pruning + Weight Sharing, and the detail results are listed in Table 4. We used prune ratio of 0.3 and 0.5, and set the number of bits after weight sharing at 4. We can see that though the accuracy decrease after applying WS, for most experiments

we can mostly recover the performance after retraining the weight shared model. When there is no teacher model, the final retrained model can even consistently outperform the original uncompressed model. The results indicate that applying weight sharing after pruning is generally effective in terms of compressing the model and at the same time keeping the performance.

Compression	Model	ResNet10	ResNet18	ResNet26
Pruning + WS Prune ratio = 0.3 compression 1.25x (29.98% pruned) WS: 4 bits	No-teacher	0.7835	0.7814	0.7803
	+ Pruning	0.7763	0.7812	0.7798
	+ Pruning Retraining	0.7886	0.7934	0.7928
	+ WS	0.7822	0.7814	0.7791
	+ WS Retraining	**0.7910**	**0.7944**	**0.7931**
	FitNet	**0.8460**	**0.8526**	0.8541
	+ Pruning	0.8260	0.8395	0.8509
	+ Pruning Retraining	0.8429	0.8518	**0.8556**
	+ WS	0.5552	0.6771	0.5761
	+ WS Retraining	0.8265	0.8400	0.8053
	HintonKD	0.6999	0.7243	0.7231
	+ Pruning	0.6824	0.7117	0.7165
	+ Pruning Retraining	**0.7523**	**0.7528**	0.7428
	+ WS	0.6839	0.7079	0.6882
	+ WS Retraining	0.7505	0.7478	**0.7448**
Pruning + WS Prune ratio = 0.5 compression 1.43x (49.98% pruned) WS: 4 bits	No-teacher	0.7835	0.7814	0.7803
	+ Pruning	0.7664	0.7710	0.7657
	+ Pruning Retraining	**0.7908**	**0.7907**	**0.7930**
	+ WS	0.7796	0.7795	0.7849
	+ WS Retraining	0.7873	0.7897	0.7876
	FitNet	**0.8460**	0.8510	**0.8567**
	+ Pruning	0.7901	0.8137	0.8237
	+ Pruning Retraining	0.8407	**0.8514**	0.8531
	+ WS	0.3763	0.3892	0.4749
	+ WS Retraining	0.8142	0.8302	0.8259
	HintonKD	0.7005	0.7224	0.7232
	+ Pruning	0.6512	0.6923	0.6489
	+ Pruning Retraining	0.7476	**0.7497**	**0.7459**
	+ WS	0.7111	0.6626	0.6243
	+ WS Retraining	**0.7490**	0.7471	0.7405

Table 4: Experimental result by applying pruning and weight sharing techniques.

5 Conclusion

In this work, we examined the effectiveness when combining various Knowledge Distillation methods with other different model compression techniques, including pruning, weight sharing and quantization. We perform comprehensive experiments on CIFAR10 with a few variants of ResNets. Here is a summary of the key observations: (a) Pruning with retraining works well for model trained with or without KD. (b) Weight sharing with 4 bits per layer can generally achieve good compression rate without hurting performance, but it does not work well with HintonKD. (c) QAT can increase the performance when using HintonKD, but will hurt the accuracy when applied with FitNets especially when the model size is larger. (d) Pruning after QAT is generally safe in terms of compression without losing performance. (e) Pruning followed by weight sharing can effectively reduce model size while keeping the same level of accuracy. We hope these discoveries can provide guidance for model compression in the industry to produce efficient light-weight models without compromising performance.

References

[1] Yinpeng Dong, Renkun Ni, Jianguo Li, Yurong Chen, Jun Zhu, and Hang Su. Learning accurate low-bit deep neural networks with stochastic quantization. *arXiv preprint arXiv:1708.01001*, 2017.

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

[3] 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.

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

[5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[6] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[7] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[8] Akshay Kulkarni, Navid Panchi, Sharath Chandra Raparthy, and Shital Chiddarwar. Data efficient stagewise knowledge distillation. *arXiv preprint arXiv:1911.06786*, 2019.

[9] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[10] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[11] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[12] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[13] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2082–2090, 2016.

[14] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision*, pages 365–382, 2018.

[15] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.