This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2886876, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Prune deep neural networks with the modified $L_{1/2}$ penalty

JING CHANG[1], JIN SHA[1,2]
[1]School of Electronic Science and Engineering, Nanjing University, Nanjing 210023 (e-mail: cj_nju@163.com)
[2]Shenzhen Research Institute, Nanjing University, China (e-mail: shajin@nju.edu.cn)

Corresponding author: Jin Sha (e-mail: shajin@nju.edu.cn).

This work was supported by the National Key R&D Program of China (No.2016-YFA0202102), the National Natural Science Foundation of China under Grant No. 61370040, the project on the Industry Key Technologies of Jiangsu Province, BE2017153, Shenzhen science and technology project JCYJ20170818110714340, the Fundamental Research Funds for the Central Universities 021014380084, 021014380063 and a Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD).

ABSTRACT Demands to deploy deep neural network (DNN) models on mobile devices and embedded systems have drastically grown in recent years. When transplanting DNN models to such platforms, requirements pertaining to computation and memory use are bottlenecks. To overcome them, network pruning has been carefully studied as a method of network compression. The effectiveness of network pruning is significantly affected by incorrect pruning on important connections. In this paper, we propose a network pruning method based on the modified $L_{1/2}$ penalty, that reduces incorrect pruning by increasing the sparsity of the pretrained models. The modified $L_{1/2}$ penalty yields better sparsity than the L_1 penalty at a similar computational cost. Compared with past work that numerically defines the importance of connections and re-establishes important weights when incorrect pruning occurs, our method achieves faster convergence by using a simpler pruning strategy. The results of experiments show that our method can compress LeNet300-100, LeNet-5, ResNet, AlexNet and VGG16 by factors of 66×, 322×, 26×, 21×, and 16×, respectively, with negligible loss of accuracy.

INDEX TERMS Deep learning, Loss function, Neural Network, Network Pruning

I. INTRODUCTION

IN the past decade, deep learning has made breakthroughs in many important applications, such as object classification [1], speech recognition [2] and natural language processing [3]. With the increasing size of datasets and the evolution of deep learning models, designing deeper and larger models has become one of the main trends to further exploit the capability of machine learning. Deeper and larger models usually have more parameters and more complex network structures. More parameters imply larger storage requirements and more memory access, which drastically increase energy consumption. Moreover, this trend increases the requirements for computational capacity and memory bandwidth. Research on deep learning could not have advanced without the development of graphics processing units(GPU), Field Programmable Gate Array(FPGA) and Application Specific Integrated Circuit(ASIC,such as TPU [4]) also play important roles in forming inferences in deep learning. With the growing need to deploy deep learning models in production, DNN models have begun being widely implemented on mobile devices and embedded systems, where hardware resources and battery capacity are strictly constrained. The contradiction between the growing complexity of deep learning models and limitations on application platforms is a fundamental problem. Cloud computing can share part of the burden, but in applications like autonomous driving, where communication delay is a critical issue, local computing remains the only choice. Therefore, network compression is essential to make DNN models friendlier to these platforms.

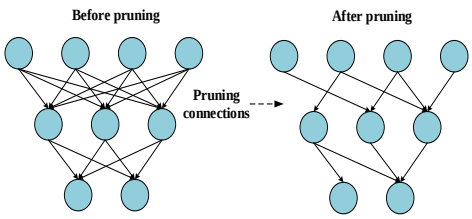


FIGURE 1. DNN model before and after pruning

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2886876, IEEE Access

Although DNN models normally require a large number of parameters to guarantee good performance, significant number of redundancies reside in these parameters [5]. By using a suitable method, DNN models can be compressed into smaller models without a noticeable loss in predictive accuracy. Of the available methods, network pruning is outstanding owing to its impressive capability to simultaneously obtain a high compression rate and negligible loss of accuracy. DNN models are pruned by removing parts of their connections or neurons to obtain sparser network structures with fewer parameters, as shown in Figure 1. Network pruning reduces requirements pertaining to both storage and memory access. As fetching data from DRAM is very energy inefficient, network pruning also drastically reduces energy consumption. With customized sparse matrix accelerators [6], the pruned models can run fast on limited hardware resources.

Network pruning is usually applied to pretrained networks, such as the pruning method proposed by Han et al. [7]. When this magnitude-based pruning is applied to a DNN model, the weights close to zero usually have little effect on the outputs of the corresponding neurons, thus these weights are considered unimportant and are removed. However, this method inevitably suffers from incorrect pruning, where important weights with relatively small magnitudes are mistakenly removed. To reduce the negative effects of incorrect pruning, Guo et al. [8] incorporated the splicing operation into network pruning. Based on numerically predefined importance of connections, the splicing operation enables connection recovery as long as the pruned connections are found to be important during training. This method improves compression rates without significant loss of accuracy, but its training algorithm is complicated and suffers from slow convergence. Therefore, we seek to avoid incorrect pruning from another point of view. The intuition is that if there is significant numerical divergence between important and unimportant weights, the negative impact on the capacity of DNN models caused by incorrect pruning will be reduced. This implies that increasing the sparsity of pretrained networks is a promising means of avoiding incorrect pruning. Usually, L_1 regularization is used to increase the sparsity of models.

In this paper, the modified $L_{1/2}$ penalty, which yields better sparsity than L_1, is introduced to increase the sparsity of DNN models. Empirical results indicate that incorrect pruning can be effectively avoided. Following a similar pruning pipeline proposed by Han et al. [7], we compressed the number of parameters in LeNet300-100, LeNet-5, ResNet, AlexNet and VGG16 by a factor of 66×, 322×, 26×, 21×, and 16×, respectively, where our method outperformed other network pruning methods. Furthermore, the modified $L_{1/2}$ penalty can cooperate with other compression strategies to achieve higher compression rates.

The remainder of this paper is organized as follows: In Section II, related work is introduced briefly. In Section III, the $L_{1/2}$ penalty and its modification are presented. Section IV details an experimental analysis of the proposed method, and the conclusions of this paper are drawn in Section V.

II. RELATED WORK

Network compression is an area of active research. Many methods have been proposed to apply DNNs to mobile devices, and can be roughly classified into four categories: weight precision reduction, matrix or tensor decomposition, vector quantization and network pruning.

A. REDUCTION IN WEIGHT PRECISION

Vanhoucke et al. [9] explored a fixed-point DNN implementation with eight-bit integer activations instead of 32-bit floating point. Several groups [10] have used this idea and discovered that reducing the encoding precision of weights has litte effect on the loss in accuracy during both training and inference. Furthermore, some researchers [11], [12] have applied binary quantization to DNN models and constrained the weights to +1 and -1. Although these methods do compress DNN models to some extent by reducing the precision of weights, their compression rates are limited and memory bandwidth remains the bottleneck.

B. MATRIX OR TENSOR DECOMPOSITION

Denil et al. [5] constructed low-rank decompositions to approximate parameter matrices. Their method achieved a 1.6× speedup on convolutional layers with a drop of 1% in prediction accuracy. Some improved methods based on this idea can provide more promising speedups [13]–[15]. Although matrix or tensor decomposition is useful for DNN compression and speedup, these methods normally incur severe losses in accuracy under high compression rates.

C. VECTOR QUANTIZATION

Gong et al. [16] explored several similar methods and emphasized the effectiveness of product quantization. HashNet proposed in [17] compresses networks by grouping their parameters into hash buckets before training. The network is trained with a standard backpropagation procedure and can achieve substantial savings in terms of storage. Han et al. [7] proposed that weights can be grouped by absolute value using k-means clustering. Ullrich et al. [18] proposed soft weight sharing that clusters weights using a mixed Gaussian distribution. Vector quantization can effectively reduce the diversity of parameters at the cost of increasing the number of indices.

D. NETWORK PRUNING

Network pruning was introduced by LeCun et al. [19]. It exploits the second derivatives of the loss function to balance training loss with model complexity. Based on this paper, Hassibi [20] proposed considering non-diagonal elements of the Hessian matrix to reduce the loss in accuracy while keeping the compression rates constant. These two methods suffer from high computational complexity when tackling large networks in spite of their theoretical optimization. Han et

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2886876, IEEE Access

al. [21] [7] explored a magnitude-based, pretraining, pruning and retraining pipeline, and reported promising compression results without a loss in accuracy. To avoid incorrectly pruning important connections when applying this method, Guo et al. [8] numerically defined the importance of connections, and re-established important weights when incorrect pruning occurred. Wen et al. [22] introduced group Lasso to the DNN loss function to obtain structural sparsity, which can accelerate computation.

The above approximation and quantization techniques mentioned above are orthogonal to network pruning, and can be used together to obtain further gains [7].

III. PROPOSED PRUNING METHOD

A. $L_{1/2}$ PENALTY

For a DNN model f, given a data set $(x_i, y_i)_{i=1}^n$, the training process can be viewed as an optimization problem:

$$\min\{\frac{1}{n}\sum_{i=1}^{n} l(y_i, f(x_i)) + \lambda\|f\|_k\} \tag{1}$$

where $l(.,.)$ is the classification loss, that usually involves cross entropy to solve multi-classification tasks. λ is the regularization parameter, and $\|f\|_k$ is the weight penalty or regularization function, normally taken as the matrix norms. Most common regularization methods can be represented as L_q, $q \geq 0$; when $q = 1$, it is L_1 or the Lasso; when $q = 2$, it is L_2 or a ridge regression. L_1 and L_2 are commonly used as regularizations of DNN models to improve generalization performances. When $q > 1$, L_q cannot obtain sparse solutions. L_1 has a certain degree of sparsity. L_0, which is constrained by the number of nonzero parameters, yields the sparsest solutions, but faces the problem of combinatorial optimization. $L_q(0 < q < 1)$ is a nature try to achieve sparser solutions than L1 while maintaining the low complexity of the optimization scheme.

By taking phase diagram studies with a set of experiments implemented on signal recovery and error correction problems, the sparsity of L_q is experimentally evaluated in [23]. The experimental results of phase diagrams are shown in Figure 2, and the higher percentage represents better sparsity. As shown in Figure 2, the ratio changes very slowly between $q \in (0, 0.5)$, while the ratio changes rapidly when q increases from 0.5 to 1. This reveals that $L_{1/2}$ regularization is significantly better than L_1 regularization, while it takes no significant difference from other L_q regularizations when $q \in (0, 0.5)$. Thus, $L_{1/2}$ can be regarded as the representative of L_q (0 < q < 1) and has good sparsity.

The mathematical expression of the $L_{1/2}$ penalty is $\sum_{i=1}^{n} \sqrt{|w_i|}$, where w denotes weight. It is a nonconvex penalty originally proposed to solve the problems of high-dimensional and massive data analysis. The $L_{1/2}$ penalty has some promising properties, such as unbiasedness, sparsity and oracle property [24]. The sparsity of the $L_{1/2}$ penalty can be understood from the perspective of geometry. Figure 3 shows the graphics of the $L_{1/2}$, L_1 and L_2 regularizations.

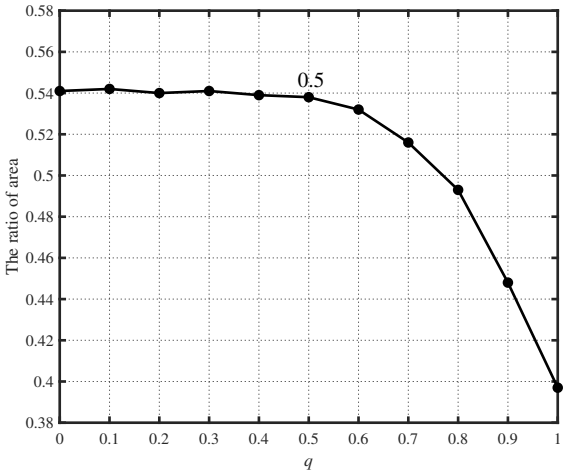
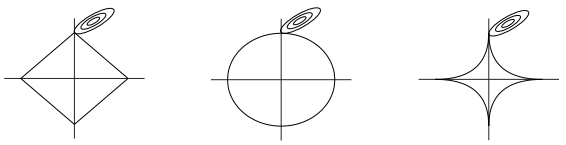


FIGURE 2. Success recovery percentage of Lq regularizations when applied to signal recovery

The contours in Figure 3 represent the solutions of classification loss functions.



(a) L_1 (b) L_2 (c) $L_{1/2}$

FIGURE 3. Estimation pictures of (a)L_1 (b)L_2 (c)$L_{1/2}$

As shown in Figure 3, the constraint region of L_1 is a rotated square. The Lasso solution is the first place where the contours meet the square, and the corners on the axis are more likely to be the points of intersection. This indicates that parameters tend to locate around zero. The graph of L_2 is shown in Figure 3(b). There is no corner for the contours to meet and, as a result, zero solutions rarely appear. It is clear that the solution to $L_{1/2}$ is more likely to occur at the corners than the solution to L_1 owing to its sharper corners, which indicates that sparser solutions can be obtained in comparison with the L_1 penalty.

Therefore, after incorporating the $L_{1/2}$ penalty into the DNN loss functions with a proper regularization parameter, sparser weight distributions of DNN models can be achieved by using gradient descent.

B. THE MODIFIED $L_{1/2}$ PENALTY

If DNN models are directly trained with the $L_{1/2}$ penalty, some problems occur. The $L_{1/2}$ penalty has the form $\sum_{i=1}^{n} \sqrt{|w_i|}$. For w_i, the gradient is $\frac{1}{2\sqrt{|w_i|}}$. When w is close to zero, its gradient is very large. This causes fluctuations in the course of training, and makes it challenging to converge

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2886876, IEEE Access

to the optimal state. Moreover, weights with values of zero cause the dividing-by-zero error. To deal with these problems, the original $L_{1/2}$ penalty is modified as follows:

$$modified\ \ L_{1/2} = \sum_{i=1}^{n} F(w_i) \tag{2}$$

where

$$F(w_i) = \begin{cases} \sqrt{|w_i|} & |w_i| \geq c, \\ \beta w_i^2 & |w_i| < c. \end{cases} \tag{3}$$

c is a positive constant close to zero, and it is set to 0.05, which is decided by the convergence results of DNN models. β is a coefficient to ensure the continuity of the gradient.

$F(w_i)$ is identical to L_2 when the absolute value of w_i is smaller than c, and L_2 is a convex function where small weights in this range are damped. By modifying the $L_{1/2}$ penalty with this technique, we constrain the weights causing fluctuations in a small neighborhood of zero. This accelerates convergence and avoids error due to divison by zero. The graph of the modified $L_{1/2}$ penalty is shown in Figure 4.

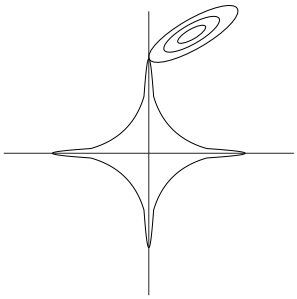


FIGURE 4. Estimation picture of the modified $L_{1/2}$ penalty

The differences between the modified $L_{1/2}$ penalty and the original $L_{1/2}$ penalty in the graphs are in parts close to the corners. Such differences reduce the likehood that the solution occurs at the corners. The negative effects on the solutions are limited when c is small. The sparsity of the modified $L_{1/2}$ penalty is experimentally investigated in Section IV.

C. PRUNING WITH THE MODIFIED $L_{1/2}$ PENALTY

Inspired by Han [7], we use a similar pruning pipeline. In the first step, the loss function is defined as follows:

$$\min\{\frac{1}{n}\sum_{i=1}^{n} l(y_i, f(x_i)) + \lambda \cdot modified\ \ L_{1/2}\} \tag{4}$$

where $l(.,.)$ is cross entropy. The modified $L_{1/2}$ is implemented as Equations (2) and (3), and λ is the regularization parameter. The DNN model with this loss function is pretrained by the SGD. In the second step, low-weight connections are removed from the network according to the predefined threshold. In the third step, the pruned connections are frozen and the model is retrained with a decaying λ. The second and the third steps are applied in an iterative way, with regularization parameter λ decaying by a factor of 10 in each iteration. The pipline of the pruning algorithm is shown in Figure 5.

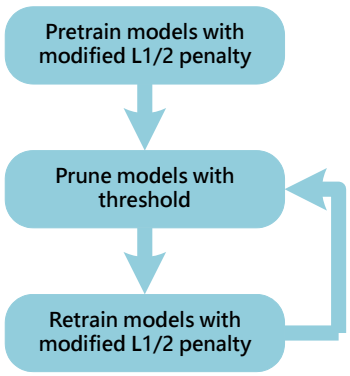


FIGURE 5. Three steps of pruning pipeline

D. ACCELERATING PRUNING PIPELINE

When applying the proposed pruning pipeline to large models like AlexNet and VGG16, the steps of both pretraining and retraining consume a long time. Two methods are used to accelerate the pruning pipeline. First, knowledge distillation is applied to both the pretraining and the retraining steps. Knowledge distillation was proposed by Hinton et al. [25] to transfer knowledge from a large and complex model to simpler models. By training a student network from the softened output of an ensemble of larger networks, teacher networks, knowledge distillation helps improve the generalization capability of simple models and accelerate convergence. Introducing knowledge distillation is an effective method to accelerate the pretraining and retraining phases.

Second, the strategy proposed by Han et al. [21] involves pruning models with more iterations instead of single-step aggressive pruning, with the aim of avoiding incorrect pruning. By benefiting from the greater sparsity implemented by the proposed method, a more aggressive strategy can be used to reduce the number of iterations. The evidence is presented in Section IV-B.

IV. EXPERIMENTS

In this section, the proposed method is experimentally analyzed. The sparsity of the modified $L_{1/2}$ penalty was first investigated, and the effect of the increased sparsity of penalties in the pretrained models of network pruning was examined. Finally, the effectiveness of our method was compared with prevalent methods.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2886876, IEEE Access

TABLE 1. Weight distributions of different penalties in LeNet-5

Penalty	$\|w\| < 0.05$	$0.05 \leq \|w\| < 0.1$	$0.1 \leq \|w\| < 0.15$	$\|w\| \geq 0.15$
None	34.09%	26.04%	17.23%	22.64%
L_2	87.20%	8.54%	2.74%	1.52%
L_1	98.09%	0.71%	0.42%	0.78%
$L_{1/2}$	98.76%	0.24%	0.20%	0.80%
modified $L_{1/2}$	98.71%	0.25%	0.21%	0.83%

A. THE SPARSITY OF THE MODIFIED $L_{1/2}$ PENALTY

To investigate the sparsity of the modified $L1/2$ penalty, several experiments are conducted on the MNIST dataset. MNIST is a database of handwritten digits and it is widely used to experimentally assess machine learning methods. The DNN model used was LeNet-5. It is a convolutional neural network(CNN) model proposed by LeCun et al. [26] for document recognition. LeNet-5 consists of two convolutional layers and two fully connected layers with 431000 trainable parameters in total.

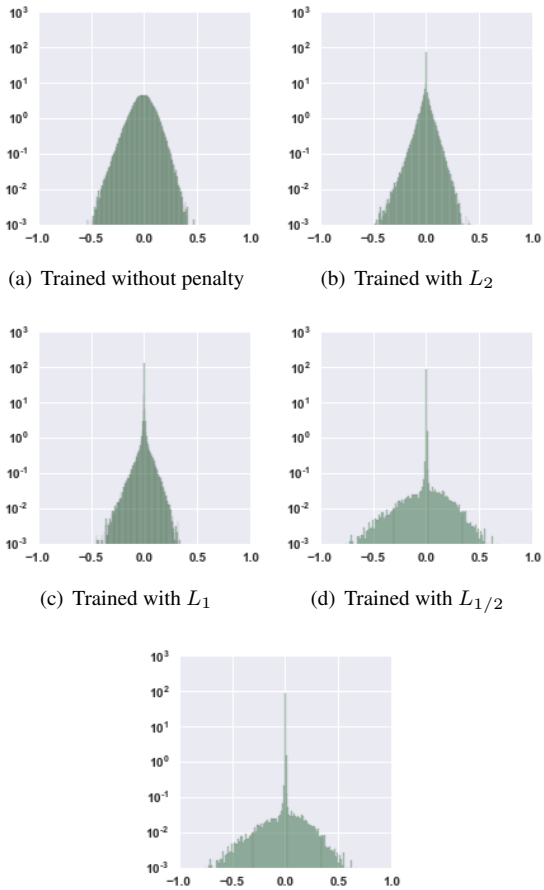
Four LeNet-5 models regularized by the original $L_{1/2}$, the modified $L_{1/2}$, L_1 and L_2 penalties were first separately are trained for 50 epochs under the same setup. A model without any penalties was also trained as a supplement. The weight distributions of these models when achieving similar test accuracies are shown in Figure 6 and Table 1.

As shown in Figure 6 and Table 1, more number of the weights in the model trained with the original $L_{1/2}$ and the modified $L_{1/2}$ penalty were limited to a small neighborhood of zero. Further, those connections with large weights were more divergent in comparison with other models. Moreover, the modified $L_{1/2}$ has similar sparsity and converges faster, compared with the original $L_{1/2}$. These results suggest that the model trained with the modified $L_{1/2}$ penalty was better in terms of sparsity.

B. HOW THE SPARSITY OF PRETRAINED MODELS AFFECTS NETWORK PRUNING

Network pruning performance is significantly degraded by incorrectly pruned connections. We experimentally investigated whether a sparser pretrained model trained with the modified $L_{1/2}$ penalty can prevent important connections from being pruned. As in the previous experiment, four models with different penalties were trained. The results of Section IV-A indicate that the model trained with the modified $L_{1/2}$ penalty had the best sparsity followed by the one trained with L_1, and the other two models had poor sparsity. Owing to the different weight distributions of the four models, pruning these models with identical thresholds was not appropriate. They are thus pruned with the identical ratio of connections, and their test accuracies before retraining are shown in Figure 7.

As shown in Figure 7, the model trained with the modified $L_{1/2}$ penalty was remarkably adaptable to aggressive pruning while the accuracies of other models incurred sharp declines. These models were then retrained without any penalty to measure the negative impact of incorrect pruning. The test accuracies are shown in Figure 8.



(a) Trained without penalty (b) Trained with L_2

(c) Trained with L_1 (d) Trained with $L_{1/2}$

(e) Trained with the modified $L_{1/2}$

FIGURE 6. Weight distributions of different penalties in LeNet-5

In Figure 8, models which with better sparsity incur smaller losses in accuracy after retraining. These results empirically prove the hypothesis that incorrect pruning on important connections can be reduced by increasing the sparsity of the pretrained models. They also indicate that pretraining models with the modified $L_{1/2}$ penalty is an effective method to reduce the negative impact of incorrect pruning.

As these experiments illustrate, the proposed pruning method yields higher accuracy when pruned aggressively. Therefore, a more aggressive pruning strategy can be used

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2886876, IEEE Access

TABLE 2. Compression Results

Model	Parameters	Method	Top1 error rate	Params(%)
LeNet300-100	267K	Pruning pipeline	1.64%→1.58%	8% (× 12)
		Dynamic surgery	2.28%→1.99%	1.8% (× 56)
		Soft weight sharing	1.89%→1.94%	4.3% (× 23)
		Ours	1.80%→1.83%	**1.5% (× 66)**
LeNet-5	431K	Pruning pipeline	0.80%→0.74%	8% (× 12)
		Dynamic surgery	0.91%→0.91%	0.9% (× 108)
		Soft weight sharing	0.88%→0.97%	0.5% (× 200)
		Ours	0.90%→0.95%	**0.31% (× 322)**
Res-Net	2.7M	Soft weight sharing	6.48%→8.50%	6.6% (× 15)
		Ours	7.40%→8.20%	**3.8% (× 26)**
AlexNet	61M	Pruning pipeline	42.78%→42.77%	11% (× 9)
		Dynamic surgery	43.42%→43.09%	5.7% (× 17.7)
		Ours	42.94%→43.06%	**4.8% (× 21)**
VGG16	138M	Pruning pipeline	31.50%→31.34%	7.5% (× 13)
		Ours	31.77%→31.80%	**6.4% (× 16)**

TABLE 3. LeNet Results

Model	Layers	Params.	Params(%) [7]	Params(%) [8]	Params(%)(Ours)
LeNet5	conv1	0.5K	66%	14.2%	15.4%
	conv2	25K	12%	3.1%	1.9%
	fc1	400K	8%	0.7%	0.15%
	fc2	5K	19%	4.3%	3.8%
	Total	431K	8%	0.9%	**0.31%**
LeNet300-100	fc1	236K	8%	1.8%	1.5%
	fc2	30K	9%	1.8%	2.4%
	fc3	1K	26%	5.5%	7.9%
	Total	267K	8%	1.8%	**1.5%**

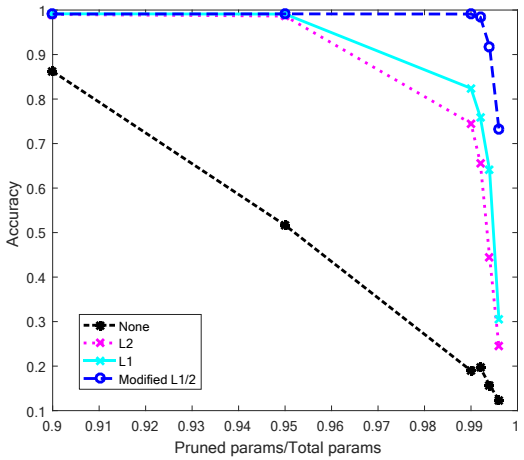


FIGURE 7. Test accuracy of models pruned by identical ratios before retraining

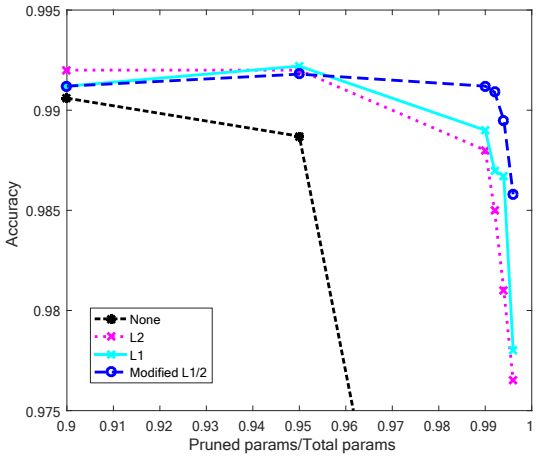


FIGURE 8. Test accuracy of models pruned by identical ratios after retraining

to accelerate the pruning pipeline.

In terms of computational cost, the time for one iteration was nearly the same, as shown in Table 4, when models were trained with the modified $L_{1/2}$, L_1, and L_2 penalties separately. Therefore, the modified $L_{1/2}$ penalty achieved better sparsity without incurring extra computational cost.

C. COMPARISON WITH THE EXISTING NETWORK PRUNING METHODS

To compare with the prevalent network pruning methods, the proposed method was applied to some popular DNN models, including LeNet-5 and LeNet300-100, on the MNIST dataset, a later version of ResNet proposed by Zagoruyko [27] on CIFAR-10, and AlexNet and VGG16 on ImageNet. All these models were implemented on TensorFlow framework with default experimental settings for the SGD method, including the base learning rate,training

TABLE 4. The comparisons of computation time for different penalties

Model	Batch size	Penalties	Time per epoch(second)
LeNet300-100	256	None	2s
	256	L2	2s
	256	L1	2s
	256	Modified $L_{1/2}$	2s
LeNet-5	256	None	4s
	256	L2	4s
	256	L1	4s
	256	Modified $L_{1/2}$	4s
Res-Net	128	None	176s
	128	L2	177s
	128	L1	176s
	128	Modified $L_{1/2}$	177s

batch size, learning policy and maximal number of training iterations. Experiments are conducted based on the pruning method presented in Section III-C, and the results are shown in Table 2. We compared our method with the pruning pipeline [7], dynamic surgery [8] and soft weight sharing [18] on LeNet300-100, LeNet-5 on MNIST, ResNet on CIFAR-10, and AlexNet and VGG16 on ImageNet. Using the satisfactory sparsity of the modified $L_{1/2}$ penalty, our method achieved the state-of-art performance on several models based on different datasets. Due to the knowledge distillation and the aggressive pruning strategy, the pretraining and retraining steps require far fewer iterations than in the method proposed by Han et al. [7].

1) LeNet on MNIST and ResNet on CIFAR-10

Experiments were first conducted on the MNIST dataset using LeNet-5 and LeNet300-100. LeNet-5 has been introduced above, LeNet300-100 is a fully connected network with two hidden layers, with 300 and 100 neurons each. Pretrained with the modified $L_{1/2}$ penalty, LeNet-5 and LeNet300-100 achieved error rates of 0.9% and 1.8% on MNIST.

Both the convolutional layers and fully connected layers were pruned with separate thresholds within one iteration. The threshold for each layer was determined in a greedy layer-by-layer way. After two iterations of the proposed pruning method, the parameters of LeNet-5 and LeNet300-100 were reduced to 0.31% and 1.5% respectively. The layer-by-layer comparisons are shown in Table 3.

Experiments were also conducted on CIFAR-10 dataset using ResNet. The parameters chosen for ResNet were depth = 16 and width = 4. Dropout was not applied in this experiment. The resultant network had 2.7M parameters. Error rates of 5.02% and 24.03% on CIFAR-10 and CIFAR-100 were obtained respectively. By re-implementing this model, the obtained error rates were 7.4% and 28.75%. We used the same pruning method on ResNet, and removed up to 96.2% of the parameters.

TABLE 5. Alexnet Results

Layers	Params.	Params(%) [7]	Params(%) [8]	Params(%)(Ours)
conv1	35K	84%	53.8%	54.3%
conv2	307K	38%	40.6%	38.4%
conv3	885K	35%	29.0%	27.1%
conv4	664K	37%	32.3%	33.1%
conv5	443K	37%	32.5%	31.6%
fc1	38M	9%	3.7%	3.6%
fc2	17M	9%	4.6%	3.3%
fc3	4M	25%	6.6%	7%
Total	61M	11%	5.7%	**4.8%**

2) Alexnet and VGG16 on ImageNet

We further examined the performance of the proposed method on the ImageNet ILSVRC-2012 dataset, which had 1.2 million training examples and 50,000 validation examples.

AlexNet, which was proposed by Krizhevsky et al. [1], and had 61 million parameters with five convolutional layers and three fully connected layers. Without any data augmentation, the pretraining model achieved top-1 error rate of 42.94%. Deep neural networks suffer from the problem of gradient vanishing [28], which makes it challenging for the pruned models to recover their accuracies. To avoid this problem, the convolutional layers and fully connected layers were pruned separately, and only one kind of layers was pruned within one iteration. The layer-by-layer comparisons are shown in Table 5. Benefiting from the pruning pipeline acceleration mentioned above, it only took 120 epochs to obtain a satisfactory pruning model. The proposed method thus has advantages in terms of both the results of pruning and convergence speed compared with the methods developed by Han et al. [7] and Guo et al. [8].

TABLE 6. VGG16 Results

Layers	Params.	Params(%) [7]	Params(%)(Ours))
conv1_1	2K	58%	64.7%
conv1_2	37K	22%	21.6%
conv2_1	74K	34%	33.7%
conv2_2	148K	36%	31.8%
conv3_1	295K	53%	54.2%
conv3_2	590K	24%	22.0%
conv3_3	590K	42%	37.2%
conv4_1	1M	32%	33.0%
conv4_2	2M	27%	24.2%
conv4_3	2M	34%	30.5%
conv5_1	2M	35%	31.7%
conv5_2	2M	29%	27.5%
conv5_3	2M	36%	29.7%
fc6	103M	4%	3.0%
fc7	17M	4%	3.4%
fc8	4M	23%	25.1%
Total	138M	7.5%	**6.4%**

VGG16 [29] is a larger and deeper model than AlexNet. It contains many more convolutional layers, composed of convolution kernels of the same size(3×3). The model pretrained with the modified $L_{1/2}$ penalty achieved top-1 error rate of 31.77%. To further reduce the impact of the vanishing gradient, the convolutional layers of VGG16 were divided into 5 blocks. Each block consisted of several convolutional

layers with feature maps of the same size, and was pruned separately when using the proposed pruning method.

The results of pruning were very promising, as shown in Table 6. Compared with the method proposed by Han et al. [7], ours improved the pruning results by reducing the negative impact of incorrect pruning. The parameters of the model were reduced in number to 6.4%, and the largest fully connected layers were pruned most significantly. Following pruning, VGG16 was much friendlier to mobile devices in light of the requirements of storage and memory access.

V. CONCLUSION

In this paper, we investigated methods to compress DNN models, and proposed one to improve the performance of network pruning by introducing the modified $L_{1/2}$ penalty in both the pretraining and the retraining procedures. Unlike past work, a simpler way was introduced to reduce incorrect pruning by increasing the sparsity of the pretrained models. Benefitting from the satisfactory sparsity of the modified $L_{1/2}$ penalty, the proposed method achieved the state-of-the-art performance. LeNet300-100, LeNet-5, ResNet, AlexNet and VGG16 are compressed by a factor of 66×, 322×, 26×, 21×, and 16× respectively with negligible loss in accuracy. The modified $L_{1/2}$ penalty is also computation friendly. Furthermore, the proposed method can be used together with other compression methods to obtain further gains.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in International Conference on Neural Information Processing Systems, pp. 1097–1105, 2012.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N. Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82–97, 2012.

[3] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, C. Potts, et al., "Recursive deep models for semantic compositionality over a sentiment treebank," in Proceedings of the conference on empirical methods in natural language processing (EMNLP), vol. 1631, p. 1642, 2013.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., "In-datacenter performance analysis of a tensor processing unit," arXiv preprint arXiv:1704.04760, 2017.

[5] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al., "Predicting parameters in deep learning," in Advances in Neural Information Processing Systems, pp. 2148–2156, 2013.

[6] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in ACM/IEEE International Symposium on Computer Architecture, pp. 243–254, 2016.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv preprint arXiv:1510.00149, 2015.

[8] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in Advances In Neural Information Processing Systems, pp. 1379–1387, 2016.

[9] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop, vol. 1, p. 4, 2011.

[10] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, "Deep neural network compression with single and multiple level quantization," arXiv preprint arXiv:1803.03289, 2018.

[11] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," arXiv preprint arXiv:1602.02830, 2016.

[12] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in Advances in Neural Information Processing Systems, pp. 3123–3131, 2015.

[13] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in Advances in Neural Information Processing Systems, pp. 1269–1277, 2014.

[14] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1984–1992, 2015.

[15] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," arXiv preprint arXiv:1412.6553, 2014.

[16] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," arXiv preprint arXiv:1412.6115, 2014.

[17] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in International Conference on Machine Learning, pp. 2285–2294, 2015.

[18] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," arXiv preprint arXiv:1702.04008, 2017.

[19] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in International Conference on Neural Information Processing Systems, pp. 598–605, 1989.

[20] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in Advances in neural information processing systems, pp. 164–171, 1993.

[21] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," pp. 1135–1143, 2015.

[22] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in Advances in Neural Information Processing Systems, pp. 2074–2082, 2016.

[23] Z. Xu, X. Chang, F. Xu, and H. Zhang, "L1/2 regularization: a thresholding representation theory and a fast solver.," IEEE Transactions on Neural Networks & Learning Systems, vol. 23, no. 7, p. 1013, 2012.

[24] J. Fan and H. Peng, "Nonconcave penalized likelihood with a diverging number of parameters," Annals of Statistics, vol. 32, no. 3, pp. 928–961, 2004.

[25] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," Computer Science, vol. 14, no. 7, pp. 38–39, 2015.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[27] S. Zagoruyko and N. Komodakis, "Wide residual networks," arXiv preprint arXiv:1605.07146, 2016.

[28] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, 2002.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Computer Science, 2014.