AUTO NETWORK COMPRESSION WITH CROSS-VALIDATION GRADIENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Network compression technology can compress large and complex networks into small networks, so that it can be deployed on devices with limited resources. Sparse regularization method, such as L^1 or L^{21} regularization, is the most popular method that can induce sparse model. However, it introduces new hyperparameters, which not only affects the degree of sparsity, but also involves whether the network can be effectively trained (gradient explosion or model non-convergence). How to select hyperparameters becomes an important and open problem for regularization-based network compression method. In this paper, we propose an auto network compression framework with cross-validation gradient which can automatically adjust the hyperparameters. Firstly, we design an unified framework which combines model parameter learning with hyperparametric learning. Secondly, in order to solve the problem of non-derivability of L^1 norm, we introduce auxiliary variables to transform it into a solvable problem, and then obtain the derivative of model parameters with respect to hyperparameters. Finally, the derivative of the hyperparametric vector is solved by the chain rule. In solving the inverse problem of Heisen matrix, we compare three methods and only calculate the mixed partial derivatives. To a certain extent, this method realizes the automatic network compression. Classical network structures such as VGG, ResNet and DensNet are tested on CIFAR-10 and CIFAR-100 datasets to prove the effectiveness of our algorithm.

1 INTRODUCTION

Network compression technology compresses the increasing size of model into one that occupies less memory resources and reduces the amount of computation of the big model, making it possible for AI to run on mobile phones. Common model compression methods are divided into four different categories: network pruning, network quantization, knowledge distillation, and matrix-based low-rank approximation (Cheng et al., 2018; Lee et al., 2019; Sakr & Shanbhag, 2019; Crowley et al., 2018; Peng et al., 2018). Pruning simplifies the large networks and select one of the subsets of large model essentially. It can reduce more than half of the parameters without significantly reducing the accuracy and greatly improve the compression rate. Our method discussed in this paper fall into the category of pruning.

Pruning based on regularization term is a very popular method. Classical machine learning methods use L^0 regularization to induce sparse models. Because L^0 norm is difficult to optimize, it is usually replaced by L^1 norm. In the field of deep learning, previous work used various norms to train the model parameters (Han et al., 2015; Mummadi et al., 2019; Liu et al., 2019; Li et al., 2019; Mehta et al., 2019). However, few work consider the problem of selecting hyper-parameters which introducd by regularization term. Hyperparameters not only determine the penalty intensity of the model, but also affect the network accuracy after fine-tuning. So how to select hyperparameters becomes an important and open problem for model compression.

In order to alleviate the above problem, a general method for automatically determining hyperparameters is studied. Both L^1 norm and L^2 norm are conducive to induce sparse model. The objective function of our study is based on the above two norms. We calculate the gradients of the coefficients of L^1 and L^2 regularization terms, and use cross validation to update hyperparameters. Our contributions are summarized as follows:

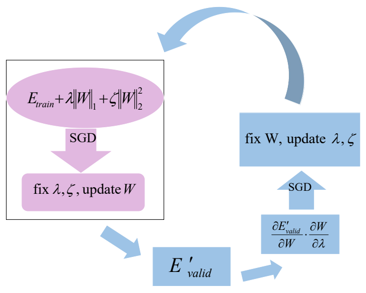
$E_{train}+\lambda\|W\|_1+\zeta\|W\|_2^2$
SGD
fix λ, ζ, update W
E'_{valid}
$\frac{\partial E'_{valid}}{\partial W}\cdot\frac{\partial W}{\partial \lambda}$
SGD
fix W, update λ, ζ

Figure 1: The whole big circle is only one iteration, and the white bottom rectangle on the left is the optimization process of the inner neural network. The inner process optimizes W by mini-batch SGD to minimize the loss function in training set. The blue part on the right is the gradient descent of cross validation loss on validation set. Weight is fixed and regularization coefficient is updated for several epoches.

- We propose a unified framework (see figure 1) that combines the whole parameters training process with hyperparametric optimization.
- Using auxiliary variables, we construct a problem that satisfies the differentiable condition and solve the derivative problem of network parameters with respect to hyperparameter. The chain rule of derivatives is used to obtain the complete solution of derivatives of validation set loss with respect to hyperparameters. Finnally, the stochastic gradient descent algorithm are used to update and optimize the hyperparameters.
- Compared with state of art methods, our algorithm achieves a higher sparse rate in VGG, ResNet, DensNet on CIFAR-10 and CIFAR-100 datasets. Accuracy loss is controlled in a negligible range.

The structure of the paper is as follows: The second part introduces the related works: regularization-based sparse method and gradient-based hyperparameter selection; the third part introduces the whole framework, focuses on the cross-validation gradient solution of non-derivative norm, and gives the corresponding theorem and proof. Finally, a hyperparameter gradient approximation method and algorithm are introduced, which effectively solves the optimization problem of deep neural network. The fourth part mainly compares with state of art methods using three different types of network models and two data sets.

2 Related Works

2.1 Deep Neural Network Pruning

The pruning methods are divided into supervised pruning and unsupervised pruning according to different criteria. Unsupervised criteria, as the name implies, analyze the importance of model parameters after training, and the analysis rules need not to be acquired through learning. For example, the L^1 norm criterion was used in Li et al. (2017) and followed by retraining. This method calculates the L^1 norm values of every channel in each layer, and retains the channel weights with large norm values. It is simple and effective. The learned rules (Liu et al., 2017; Zhuang et al., 2018; Luo et al., 2017) are more persuasive than simply determined by experience. Liu et al. (2017) learned an important factor for each channel and adds sparse constraints to the objective function. After the training is completed, the less important channels under a certain threshold are removed and retrain the new network. DCP (Zhuang et al., 2018) chose the larger F norm of the current channel gradient as the channel to be retained. Luo et al. (2017) used the output of the next layer to guide the pruning of the current layer, that is, to approximate the output with a subset of the input of the current layer. The channels only in the subset are retained. We synthesize the advantages of L^1 and L^2 norms and add them to the objective function. The difference between L^2 norm and weight delay is that L^2 norm only acts on weight coefficients. Compared with other norms, L^1 norm has better sparsity performance.

Adding sparse regularization term to the original loss function is another technical means in pruning algorithm. Han et al. (2015) proposed to use L^1 and L^2 norms to induce network sparsity, and found that L^1 regularization can achieve better accuracy after pruning, while sparse network induced by L^2 regularization can achieve higher accuracy after fine-tuning. Mummadi et al. (2019) proposed that the bounded-L^{p0} norm is used to approximate the parameters to zero to ensure the accuracy of the task, and then the bounded-L^{p0} norm is used to further induce the parameters of additional gating layer to 0. However, two regularization coefficients are directly given without further discussion. Liu et al. (2019) used L^{21} norm of each instance feature to learn sparsity during training. The selection of the coefficients of the regularization factor is only based on a limited test. Li et al. (2019) proposed a criterion to measure the importance of the channel by using the Group Lasso of two consecutive layers, and then the pruned network is trained with a OICSR loss. Mehta et al. (2019) discussed the error rate of different regularization coefficients under different optimizers. Although regularization-based pruning methods have been extensively studied, none of them explored regularization coefficients in a manner of automatically updating hyperparameters. Based on this background, we analyze how to optimize the regularized hyperparameters. As far as we know, this is the first time that the coefficient of a regular term is determined by an automatic method.

2.2 Hyperparameter Optimization based on Gradient

There are many studies on hyperparameter selection, such as grid search, random search, bayesian optimization and gradient-based cross-validation (Barratt & Sharma, 2018). The gradient-based method can find a suitable solution more quickly. Rencent works have explored gradient-based hyperparameter optimization. They focus on optimizing multiple parameters and reducing computational costs. Barratt (2018) proposed that the solution map of convex optimization problems satisfying certain conditions is differentiable by implicit function theorem. Barratt & Sharma (2018) demonstrated that the derivatives of the solution for many common convex machine learning algorithms: logistic regression, elastic-net regression, support vector machines.

In the field of deep learning, there are also some studies on gradient optimization of hyperparameter for different purposes. Ren et al. (2018) solved the problem of hyperparameter adjustment for example reweighting algorithms using cross validation based on gradient descent. Reverse mode automatic differentiation allow for optimizing hyperparameters with gradients (Maclaurin et al., 2015). But they consider that every iteration in the whole gradient descent, it is a huge overhead to store all gradients in memory, so they propose a reverse mode differentiation of SGD with finite precision arithmetic to avoid avoid the memory limitation on the task of hyperparameters optimization. Fu et al. (2016) distilled the knowledge of the gradients in the forward propagation and transfer them by adding short connections, so that the gradients in the forward propagation can be approximately calculated in the reverse process. This algorithm greatly improves computational efficiency and reduces memory consumption. Luketina et al. (2016) only considered one iteration in the training process, and approximated the Heisen matrix as a identity matrix.

These works mainly focus on the optimization of multiple parameters and only consider traditional machine learning problems or shallow neural network due to a large amount of computation and memory consumption. We mainly study the hyperparameter selection of large-scale neural network models (VGG-16, ResNet-56, ResNet-110, DenseNet-40) for the purpose of model compression. In the context of sparse regularization compression, we often encounter non-derivative cases, such as L^1 norm, L^{21} norm. Direct hyperparameter gradient is almost impossible to obtain. In this paper, we use auxiliary variables to transform non-derivable norms into equivalent derivable optimization problems. We also redesign the training process and propose a new framework which combines model parameter training with hyperparametric learning to compress big model, which makes the hyperparameter optimization of large CNN models possible and obtains sparse neural network models.

3 The Proposed Algorithm

3.1 Regularization-based Network Sparsification Method

In order to obtain sparse network structures and minimize the loss drop, we use an objective function with L^1 and L^2 regularization terms for neural networks. (see equation 1). Adding L^1 regularization term can induce sparse neuronal connections. The purpose of adding L^2 regularization term is to reduce the over-fitting of the model and increase the generalization performance of the model. Weight delay is such a technique to avoid model over-fitting. The L^2 norm used in this paper is slightly different from the weighted delay. The weighted delay is limited to all parameters, including bias terms, but the L^2 norm is only limited to the weighted coefficients.

$$e(\mathbf{W}, \lambda_1, \lambda_2) = -\sum_{n=1}^{N}\sum_{k=1}^{m}(y_k^n \log o_k^n) + \lambda_1 w_1 + \frac{\lambda_2}{2} w_2, \tag{1}$$

$$w_1 = \sum_{l=1}^{L} \| \mathbf{W}^l \|_1 = \sum_{l=1}^{L}\sum_{i,j} |w_{i,j}^l|, \tag{2}$$

$$w_2 = \sum_{l=1}^{L} \| \mathbf{W}^l \|_F^2 = \sum_{l=1}^{L}\sum_{i,j} |w_{i,j}^l|^2. \tag{3}$$

The loss function is cross entropy function $-\sum_{k=1}^{m}(y_k^n \log o_k^n)$. $\boldsymbol{o}$ is the output of a neural network model, $\boldsymbol{y}$ is the one-hot laber. w_1 is the L^1 norm sum of all weight tensors of the target model (see definition 2). w_2 is the square sum of Frobenius norm of all weight tensors (see definition 3). The training set is $\{(\boldsymbol{X}^n, \boldsymbol{y}^n)\}$ for $n = 1, 2, 3, \ldots, N$. $(\boldsymbol{X}^n, \boldsymbol{y}^n) \in \mathbb{R}^{d\times d} \times \mathbb{R}^m$. The input is a d-dimensional image matrix. Our target is minimize e on the training set. Our objective function not only guarantees the sparsity of the model, but also improves the accuracy of the model on the test set. In order to balance loss function and regularization terms, λ_1, λ_2 are introduced to represent the weight of each part. Automatic hyperparameter selection based on gradient descent method reduces the labor and time cost. We use cross-validation gradient to select appropriate hyperparameters and obtain sparse models. The cross-validation loss function used in this paper is cv. It is also possible to use other loss functions instead of cross-entropy loss function.

$$cv(\mathbf{W}, \lambda_1, \lambda_2) = -\sum_{n=1}^{M}\sum_{k=1}^{m}(\tilde{y}_k^n \log \tilde{o}_k^n). \tag{4}$$

In addition, we minimize cv on the verification set $\{(\hat{\boldsymbol{X}}^n, \hat{\boldsymbol{y}^n})\}$ for $n = 1, 2, 3, \ldots, M$. In fact, M and N represent the batch size of training set and verification set respectively.

3.2 Our overall framework

The proposed process is as follows:

- 1. Initialize hyperparameter $\boldsymbol{\lambda} = [\lambda_1, \lambda_2]$;
- 2. On the training set $\{(\boldsymbol{X}^n, \boldsymbol{y}^n)\}$, using a standard mini-batch SGD algorithm to learn the optimal $\mathbf{W}^{(t)}$:
$$\mathbf{W}^{(t)} = \arg\min_{\boldsymbol{W}} \; e(\mathbf{W}, \boldsymbol{\lambda}). \tag{5}$$
- 3. On the verification set $\{\tilde{\boldsymbol{X}}^n, \tilde{\boldsymbol{y}}^n\}$, update the $\boldsymbol{\lambda}^{(t)}$, this is also achieve by using a mini-batch SGD algorithm.
$$\boldsymbol{\lambda}^{(t)} = \boldsymbol{\lambda}^{(t-1)} - \eta \nabla_{\boldsymbol{\lambda}} cv. \tag{6}$$
- 4. Go to step 2 or stop when satisfie the termination condition of iteration (see equation 7) and obtain sparse model.

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \; cv(\mathbf{W}, \boldsymbol{\lambda}). \tag{7}$$

The key point and difficulty of the whole algorithm is to solve the hyperparameter gradient:

$$\nabla_{\boldsymbol{\lambda}} cv = \frac{\partial cv}{\partial \boldsymbol{s}(\boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} \boldsymbol{s}(\boldsymbol{\lambda}). \tag{8}$$

In equation 8, the left part derivative is the conventional derivative, while the right part derivative does not necessarily exist because that the L^1 norm is not differentiable. $\boldsymbol{s}(\boldsymbol{\lambda})$ means the intermediate variable consist of the optimal weight variables in $\mathbf{W}$. $\mathbf{W}$ can be expressed as an implicit function vector. We will give the results in the next section.

3.3 Derivatives of Solution Mapping with respect to Hyperparameters

In order to solve the problem of nondifferentiability of L^1 norm, we introduce auxiliary variables $\tilde{\mathbf{W}}^{l+} > 0, \tilde{\mathbf{W}}^{l-} > 0$ to transform the original problem into a general solvable form. Satisfy $\mathbf{W}^{l} = \mathbf{W}^{l+} - \mathbf{W}^{l-}$. $\tilde{\boldsymbol{w}}$ is a vector which consisit of the elements of $\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}$. $\boldsymbol{\mu}$ is dual variables with respect to $\mathbf{W}^{l+}, \mathbf{W}^{l-}$. $\tilde{}$ means the optimal solution.

Theorem 1. *Deep Neural Network performs classification, where $\{(\boldsymbol{X}^n, \boldsymbol{y}^n)\}$ is sample datas. The activation function is differentiable, the derivatives of the optimal solution $\tilde{\boldsymbol{w}} = [\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}]$ with respect to $\boldsymbol{\lambda}$ are:*

$$\nabla_{\boldsymbol{\lambda}} \boldsymbol{s}^*(\boldsymbol{\lambda}) = -\nabla_{\tilde{\boldsymbol{w}}} g(\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\lambda})^{-1} \nabla_{\boldsymbol{\lambda}} g(\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\lambda}). \tag{9}$$

Proof. Because L^1 is not differentiable, then we use $\mathbf{W}^{l+}, \mathbf{W}^{l-}$ ($\mathbf{W}^{l} = \mathbf{W}^{l+} - \mathbf{W}^{l-}$) to rewrite the problem as ($\mathbf{1}$ is a column vector and its elements are 1):

$$\begin{aligned} e(\mathbf{W}, \lambda_1, \lambda_2) \; &= -\sum_{n=1}^{N}\sum_{k=1}^{m}(y_k^n \log o_k^n) + \lambda_1 \sum_{l=1}^{L} \mathbf{1}^{\mathrm{T}}(\mathbf{W}^{l+} + \mathbf{W}^{l-})\mathbf{1} + \\ &+ \frac{\lambda_2}{2}\sum_{l=1}^{L}(\| \mathbf{W}^{l+} - \mathbf{W}^{l-} \|_F^2), \end{aligned} \tag{10}$$

Namely

$$\begin{array}{ll} \text{minimize} & e(\mathbf{W}, \lambda_1, \lambda_2), \\ \text{subject to} & W_{i,j,p,q}^{l+} > 0, W_{i,j,p,q}^{l-} > 0, \text{ for } l = 1, 2, 3, \ldots, L. \end{array}$$

We expansion the optimal solution $\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}$ with dual variables $\tilde{\boldsymbol{\mu}} = [\tilde{\mu_1}, \tilde{\mu_2}, \mu_3, \ldots, \mu_{d_2}]$ by vector, then we can get $\tilde{\boldsymbol{w}}$. $\boldsymbol{\theta}$ is only the optimal solution $\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}$ expansioned by vector. The dimension of $\tilde{\boldsymbol{w}}$ is $d_1 = \sum_{l=1}^{L}(4i_l j_l)$. $d_2 = 2\sum_{l=1}^{L}(i_l j_l)$ is dimension of $\boldsymbol{\mu}$. Let $\boldsymbol{s}^*(\boldsymbol{\lambda}) = \tilde{\boldsymbol{w}}^{\mathrm{T}}$ denote the optimal $\tilde{\boldsymbol{w}}$ for a given $\boldsymbol{\lambda}$ in equation 10, we can get the derivative of $\tilde{\boldsymbol{w}}$ with respect to $\boldsymbol{\lambda}$ (Barratt, 2018; Barratt & Sharma, 2018).

$$g(\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla_{\tilde{\boldsymbol{w}}^{\mathrm{T}}} e(\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\lambda}) \\ \\ -\mathbf{diag}(\tilde{\boldsymbol{\mu}})\tilde{\boldsymbol{\theta}}^{\mathrm{T}} \end{bmatrix}, \tag{11}$$

$$\nabla_{\boldsymbol{\lambda}} \boldsymbol{s}(\boldsymbol{\lambda})^* = -\nabla_{\tilde{\boldsymbol{w}}} g(\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\lambda})^{-1} \nabla_{\boldsymbol{\lambda}} g(\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\lambda}). \tag{12}$$

□

The general form of derivatives of model parameter with respect to hyperparameters is given by this theorem, which is the most important part of cross-validation gradient. The model here includes but is not limited to CNN, DNN, LSTM, RNN and other deep neural network models.

3.4 Approximate Gradient for Hyperparameter

Because the calculation of hyperparameter gradient involves the inverse operation of Heisen matrix, its complexity is cubic dimension. The order of magnitude of the parameters of the deep neural network is 10^6. The time complexity of the matrix inversion is $o(d^3) \approx o(10^{18})$. Unless the number of network parameters is small, it is impossible to calculate them. In order to calculate the inverse of Heisen matrix conveniently, we only consider the diagonal approximation of Heisen matrix. Ricotti et al. (1988) given an exact formula for calculating diagonal elements:

$$\frac{\partial^2 E_n}{\partial w_{j,i}^2} = \left[h'(a_j)^2 \sum_k \sum_k^{\prime} w_{k,j} w_{k',j} \frac{\partial^2 E_n}{\partial a_k a_k'} + h''(a_j) \sum_k w_{k,j} \frac{\partial E_n}{\partial a_k} \right] z_i^2. \tag{13}$$

Becker & Lecun (1989) and LeCun et al. (1989) given the formula for calculating the diagonal elements:

$$\frac{\partial^2 E_n}{\partial w_{j,i}^2} = \left[h'(a_j)^2 \sum_k w_{k,j}^2 \frac{\partial^2 E_n}{\partial a_k^2} + h''(a_j) \sum_k w_{k,j} \frac{\partial E_n}{\partial a_k} \right] z_i^2. \tag{14}$$

This formula is still complicated. Here we follow the example used in (Luketina et al., 2016):

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \eta_2 (\nabla_{\boldsymbol{\theta}} C_2)(\nabla_{\boldsymbol{\lambda}} \nabla_{\boldsymbol{\theta}} \tilde{C}_1), \tag{15}$$

and approximate it to a identity matrix:

$$\nabla_{\boldsymbol{\lambda}} \boldsymbol{s}(\boldsymbol{\lambda}) \quad \approx \quad -\boldsymbol{I}_d^{-1} \nabla_{\boldsymbol{\lambda}} g(\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\lambda}). \tag{16}$$

Then we can get $\nabla_{\boldsymbol{\lambda}} cv \approx -\frac{\partial cv}{\partial \boldsymbol{s}(\boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} g(\tilde{\mathbf{W}}^{l+}, \tilde{\mathbf{W}}^{l-}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\lambda})$. By simplifying the multiplication of zero

Algorithm 1 Cross-Validation Gradient Method for Model Compression

```
Input: Initialize hyperparameter $\boldsymbol{\lambda} = [\lambda_1, \lambda_2]$
 1: for all $\boldsymbol{\lambda}$ not converged do
 2:     Fix $\mathbf{W}^{(t)}$, minimize $cv$ on verification set $\{\tilde{\boldsymbol{X}}^n, \tilde{\boldsymbol{y}}^n\}$
 3:     Calculate the gradients of hyperparameters: $\nabla_{\boldsymbol{\lambda}} cv$
 4:     Update $\boldsymbol{\lambda}^{(t)} = \boldsymbol{\lambda}^{(t-1)} + \eta \frac{\partial cv}{\partial \boldsymbol{s}(\boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} g(\tilde{\mathbf{W}}^l, \boldsymbol{\lambda})$
 5:     Reinitialize parameters $\mathbf{W}^{(t)}$
 6:     for all $\mathbf{W}^{(t)}$ not converged do
 7:         Fix $\boldsymbol{\lambda}$, minimize $e$ on training set $\{(\boldsymbol{X}^n, \boldsymbol{y}^n)\}$
 8:         Update parameters $\mathbf{W}^{(t)}$ by mini-batch SGD method
 9:     end for
10: end for
```

elements, we can see that dual variables $\boldsymbol{\mu}$ can be simplified eventually. By observing the auxiliary variables in the mixed partial derivatives, their forms are consistent and can be reconstituted into one variable. Then we get:

$$\nabla_{\boldsymbol{\lambda}} cv \approx -\frac{\partial cv}{\partial \boldsymbol{s}(\boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} g(\tilde{\mathbf{W}}^l, \boldsymbol{\lambda}). \tag{17}$$

Finally, we use the automatic derivation module in PyTorch, an open source framework, to assist in calculating the hyperparameter gradient. The computational cost of the algorithm 1 is product of outer training epoches and inner training epoches.

4 Experiment

4.1 Network Model and Data Set

The network models we use are VGG (Simonyan & Zisserman, 2015), ResNet (He et al., 2016) and DenseNet (Huang et al., 2017). The vision(VGG-16) of VGG we used is consisted of 13 convolution layer, 2 full connected layer and 4 max pooling layer. The dimension of the full connection layer is 512*512 and 512*class number. We also compare residual networks with different depths of ResNet-56 and ResNet-110. DenseNet-40 is a dense connected network of 3 blocks. In each block, the features of each layer stack up and flow down to the next layer.

The data sets we use are CIFAR-10 and CIFAR-100 (Krizhevsky & Hinton, 2009). CIFAR-10 has 60,000 images: 50,000 training images and 10,000 test images. Each sample is a 32*32 color image. There are 5000 training samples in each category in CIFAR-10. The sample partitioning of CIFAR-100 is the same with CIFAR-10, but with the increase of the number of categories, the number of samples for each category is reduced to 500. This greatly improves the difficulty of classification tasks. The same model structure shows worse performance in CIFAR-10. In addition, due to the requirement of our algorithm, we divide a part of the data set from the training set as the verification set which number is 5000. Reducing the number of training samples will affect the accuracy of classification to a certain extent.

Table 1: The overall results compared with different methods on CIFAR-10. We compare four different neural network: VGG-16, ResNet-56, ResNet-110, DenseNet-40. Acc gain is the absolute error of the percentage of accuracy compared to each algorithm' own baseline for relatively fair comparison. Parameters and FLOPs represent the total parameters of the network model and the sum of addition and multiplication operations. Pruned and Reduced represent the percentage of parameter reduction and computation reduction. CV represent the proposed Cross-validation method by us.

Model	Acc gain	Parameters	Pruned	FLOPs	Reduced
VGG-16(Our baseline)	93.55%	14.99M	0%	0.31G	0%
CV-40(Ours)	**-0.26%**	**0.34M**	**97.7%**	-	-
CV-120(Ours)	**-0.21%**	**2.30M**	**84.7%**	-	-
L^1 norm (Li et al., 2017)	0.15%	5.4M	64%	0.21G	34.2%
GAL-0.05 (Lin et al., 2019)	-0.19%	3.36M	77.6%	189.49M	39.6%
GAL-0.1 (Lin et al., 2019)	-0.54%	2.67M	82.2%	171.89M	45.2%
SSS* (Huang & Wang, 2018)(GAL)	-0.33%	4.99M	66.7%	199.93M	36.3%
SSS* (Huang & Wang, 2018)(GAL)	-0.94%	3.93M	73.8%	183.13M	41.6%
COP (Wang et al., 2019)	-0.25%	1.44M	92.8%	211.2M	73.5%
ResNet-56(Our baseline)	93.25%	0.85M	0%	0.25G	0%
CV-40(Ours)	**-0.03%**	**0.34M**	**60.0%**	-	-
CV-80(Ours)	**+0.26%**	**0.79M**	**7.0%**	-	-
L^1 norm (Li et al., 2017)	0.02%	0.73M	13.7%	90.9M	27.6%
GAL-0.6 (Lin et al., 2019)	0.12%	0.75M	11.8%	78.30M	37.6%
GAL-0.8 (Lin et al., 2019)	-1.68%	0.29M	65.9%	49.99M	60.2%
NISP (Yu et al., 2018)	-0.03%	0.49M	42.60%	70.49M	43.61%
CP (He et al., 2017)	-1.0%	-	-	62.5M	50%
ADC (He & Han, 2018)	-0.9%	-	-	62.5M	50%
PP-1 (Singh et al., 2019)	-0.03%	0.06M	92.5%	21.5M	82.8%
PP-2 (Singh et al., 2019)	-0.14%	0.05M	94.3%	19.4M	84.5%
ResNet-110(Our baseline)	94.07%	1.73M	0%	0.51G	0%
CV-40(Ours)	**0.16%**	**0.47M**	**72.8%**	-	-
L^1 norm (Li et al., 2017)	-0.23%	1.16M	32.4%	155M	38.6%
GAL-0.1 (Lin et al., 2019)	0.09%	1.65M	4.1%	205.7M	18.7%
GAL-0.5 (Lin et al., 2019)	-0.76%	0.95M	44.8%	130.2M	48.5%
NISP (Yu et al., 2018)	-0.18%	0.98M	43.25%	142M	43.78%
DenseNet-40(Our baseline)	94.35%	1.07M	0%	0.57G	0%
CV-40(Ours)	**-0.23%**	**0.31M**	**71.3%**	-	-
CV-40-lr(Ours)	**-0.62%**	**0.17M**	**84.2%**	-	-
GAL-0.05 (Lin et al., 2019)	-0.31%	0.45M	56.7%	128.11M	54.7%
GAL-0.1 (Lin et al., 2019)	-1.58%	0.26M	75.0%	80.89M	71.4%
network slimming (Liu et al., 2017)	0.92%	0.66M	35.7%	381M	28.4%
network slimming (Liu et al., 2017)	0.92%	0.35M	65.2%	240M	55.0%
Synapse Pruning (Lin et al., 2018)	-0.34%	0.21M	80.4%	71M	74.82%

4.2 Experimental Environment and Configuration

Our experimental implementation is based on PyTorch with a Tesla P100 GPU. The training and sparsification of the model are accomplished together, and it is an end-to-end process. Except for the coefficients of regularization terms, the other hyperparameters are fixed. The batch sizes of training and testing are 64 and 256 respectively. The SGD momentum is 0.9. The initial learning rate is 0.1 and then divided by 10 at 1/2 and 3/4 total epoches. The weight decay is not set because that we add an L^2 regularization term only on weight. Other hyperparameters affecting the algorithm are demonstrated and analyzed in experiments.

Table 2: Ablation experiment for different start epoch of pruning on CIFAR-10 with VGG-16. Start Epoch means the epoch that begin to prune. Iter means one alternate training in training set and test set. P Paras and P rate mean the number of pruned parameters in the iteration and the percentage of pruned parameter. Acc means accuracy rate.

Start Epoch	Iter	P Paras	P rate	Test Acc	Valid Acc
10 epoch	1	6.19M	41.3%	91.02%	90.78%
	2	12.74M	85.0%	91.83%	91.60%
	3	13.63M	90.9%	92.48%	91.68%
	4	13.95M	**93.1%**	**92.53%**	**91.80%**
20 epoch	1	2.88M	19.2%	91.01%	91.32%
	2	8.65M	57.7%	91.90%	91.86%
	3	11.31M	75.5%	92.49%	91.88%
	4	12.18M	**81.3%**	**92.60%**	**92.10%**
30 epoch	1	2.13M	14.2%	91.10%	90.66%
	2	8.55M	57.0%	92.20%	91.70%
	3	11.28M	75.3%	92.11%	92.30%
	4	12.19M	**81.3%**	**92.61%**	**92.64%**
baseline	1	0M	0%	91.01%	91.18%
	2	0M	0%	91.98%	91.98%
	3	0M	0%	92.55%	92.16%
	4	0M	**0%**	**92.42%**	**92.00%**

Table 3: Ablation experiment for different number of start pruning epoches on CIFAR-10 with ResNet-56 and ResNet-110. Model-Start Epoch means model and the number of start pruning epoches with the same meaning as table 2. -B means the baseline without pruning. P Paras and P rate means the number of the pruned parameter and the the percentage of pruned parameter. Test Acc means the accuracy rate.

Model-Start Epoch	P Paras	P rate	Test Acc
ResNet-56-10	0.27M	32.2%	92.39%
ResNet-56-B	0	0%	92.58%
ResNet-110-10	0.72M	41.4%	92.70%
ResNet-110-B	0	0%	92.57%

4.3 Experimental results and analysis

Our algorithm is an automatic pruning method. The L^1 regularization term is used to induce the sparse network. When the training stage is completed, the pruning network can be obtained without fine-tuning stage. The whole framework is an end-to-end iterative process. The training and validation stages alternate to achieve faster convergence results. In the training set, the objective function is the cross-entropy loss function and two regularization terms. The cross-entropy function is to better fit the probability distribution of the training set and achieve higher accuracy. The L^2 regularization term is to constantly punish the model to improve the generalization performance of the model in the test set. The L^1 regularization term shrinks the weight coefficients to a small numerical range and tends to zero. Because L^1 is not differentiable at zero, the approximate gradient calculation can not make the weight zero in the limited number of training rounds. We design a threshold to assist the L^1 regularization term. On the verification set, we still choose the cross-entropy function as the verification loss. At this time, the parameters of the model are fixed and do not participate in the update. We calculated the gradients of L^1 and L^2 regularization coefficients. Because the number of samples in the validation set is still large, we adopted mini-batch SGD to update the super-parameters. The sample size of the verification set is only 1/10 of the training set, and the number of training rounds of each iteration is also 1/10 of the training set for empirical consideration. The above process is called an iteration. Strictly speaking, every iteration should train the model from scratch on training set. Because a complete training takes a long time, we take a roundabout way that, each iteration does not need to re-initialize the weight, but re-train the network of existing parameters. The training epoches in each iteration is regard as the total number of training epoches.

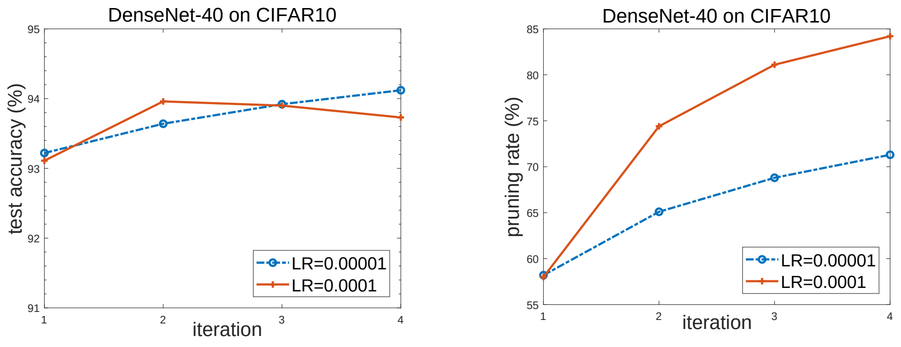

Figure 2: This Figure shows the performance of DenseNet-40 on CIFAR-10 with different learning rate of hyperparameters. We only list the data at the end of each iteration. Left: With the LR=0.00001, the test accuracy rate of DenseNet-40 is steady increasing. When the learning rate increases, there is a downward trend in test accuracy rate. Right: The larger the learning rate of hyperparameters is, the sparser the model obtained in the same iteration process is.

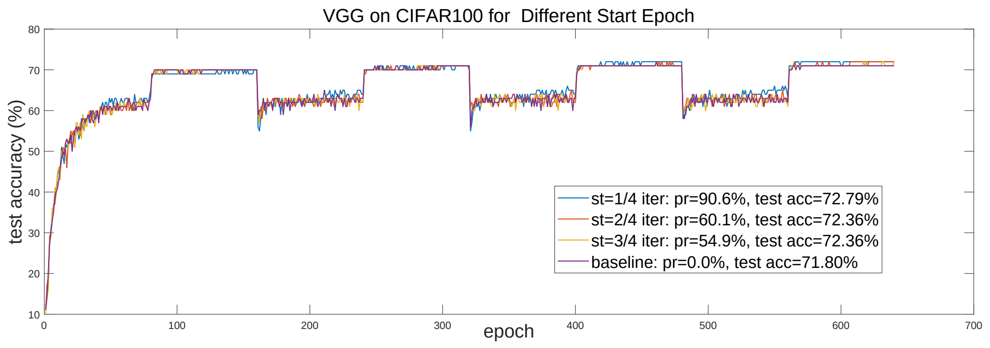

Figure 3: The performance of VGG-16 on CIFAR-100 with different starting pruning epoch. ”st” means starting pruning epoch and the fraction (such as 1/4, 2/4) means the ratio of the initial pruning epoch to the total epoch in an iteration. ”iter” meaans one iteration. ”pr” is the pruning rate and ”test acc” means the accuracy rate in test set.

Results on CIFAR-10 We compare our results with state-of-art methods in Table 1. For VGG-16, our algorithm(CV-40) achieves the highest compression rate of 97.7% with only 0.26% accuracy loss, which is higher than COP in prune rate and 0.01 lower than COP in accuracy. SSS* means it is reimplemented by GAL (Lin et al., 2019). Other results in Table 1 are copy from their original paper. ”-40” means the begining of prune epoch is 40 in each iteration and the total epoch number in each iteration is 160. Compared with GAL-0.1, CV-80 achieves few parameters of 2.3M which is less than 2.67M achieved by GAL-0.1. At the same time, our accuracy is 0.33% higher than GAL-0.1. The total computation of our algorithm is not reduced, but many parameters of the filter are changed to zero in the process of training. For ResNet-56, our algorithm achieves the same accuracy loss compared with NISP, however we have a higher compression rate of 60% than that of 42.6%. Compared with GAL-0.6 and L^1 norm, our accuracy is improved by 0.26 in the case of low compression rate. For deeper ResNet-110, we achieve a the best performance to the extent we know, the compression ratio is 72.8%. However, the accuracy rate is 0.16 higher than the baseline. The configuration of our algorithm is exactly the same with the baseline, the only difference is whether to prune or not. For DenseNet-40, compared with Synapse Pruning, we achieve a higher pruning rate with accuracy rate loss of 0.62% (CV-40-lr). ”CV-40-lr” means the starting pruning epoch is 40 with learning rate of 10^{-4}. Compared to GAL-0.05, we get a high pruning rate of 71.3% than 56.7% and a lower accuracy loss of 0.23% than 0.31%.

Ablation Experiment for the Start Prune Epoch Table 2 show the differences in different start prune epoch. The pruning threshold is 0.0001, and the weight below this threshold will be reset to

0. We divide the total training epoches by four, - x means the number of epoches that start pruning. Observed by us, the earlier the pruning starts, the better the sparsity of the network structure is. Surprisingly, the pruned network can achieves better accuracy than the baseline. The essence of pruning is to select a better network structure suitable for the task, and its performance will not decline or even improve without over-pruning. Similar results are also found on deeper residual networks, as shown in Table 3. Each iteration implies a hyperparameter update. The total number of epoch is 160 with each iteration has only 40 epoches for quickly trial run.

Ablation Experiment for the Learning Rate of Hyperparameter The learning rate of hyperparameters has a great influence on the sparsity and prediction accuracy of the network (see figure 2). Through experiments, we find that simple structure, easy-to-train network such as VGG can use the learning rate of 10^{-4} appropriately, while for densely connected composite structures such as DenseNet, a smaller learning rate of 10^{-5} can make the model still be fully trained in the process of sparse.

Results on CIFAR-100 As shown in figure 3, we compare the accuracy of VGG-16 on CIFAR-100 before and after compression. Intuitively, without considering pruning rate, the performances of models under different settings on the test set have very little difference. From the view of data, pruning does not reduce the accuracy rate, and to some extent improves the generalization of the model. Among them, the blue line is slightly higher and achieves the best performance of 72.79%. It is also the blue line that achieves the highest compression rate of 90.6%.

5 Conclusion

In order to solve the problem of setting hyperparameters in model compression algorithm based on regularization term, we propose a gradient-based hyperparameter learning method. In order to calculate the derivative of network parameters with respect to hyperparameter, we introduce auxiliary variables to construct a solvable problem. With the help of the chain rule, the derivative of verification loss with resprct to hyperparameter can be solved. This method solves two problems, one is the sparsity of deep neural network, and the other is the solution of regularization coefficient. We calculate the exact gradient of L^1 and L^2 regularization coefficients, and propose an approximate calculation method to realize the hyperparameter optimization of large convolutional neural networks. In addition, we have done comparative experiments on several classical network structures and data sets to demonstrate the effectiveness of our algorithm. In the future, we will investigate the effects of other hyperparameters such as batch size and momentum coefficient which may also affect the sparsity.

References

Shane Barratt. On the differentiability of the solution to convex optimization problems. *CoRR*, abs/1804.05098, 2018.

Shane Barratt and Rishi Sharma. Optimizing for generalization in machine learning with cross-validation gradients. *CoRR*, abs/1805.07072, 2018.

Suzanna Becker and Yann Lecun. Improving the convergence of back-propagation learning with second-order methods. 01 1989.

Jian Cheng, Peisong Wang, Gang Li, Qinghao Hu, and Hanqing Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of IT & EE*, 19(1):64–77, 2018.

Elliot J. Crowley, Gavin Gray, and Amos J. Storkey. Moonshine: Distilling with cheap convolutions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 2893–2903, 2018.

Jie Fu, Hongyin Luo, Jiashi Feng, Kian Hsiang Low, and Tat-Seng Chua. Drmad: Distilling reverse-mode automatic differentiation for optimizing hyperparameters of deep neural networks. In Subbarao Kambhampati (ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, pp. 1469–1475. IJCAI/AAAI Press, 2016.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 1135–1143. Curran Associates, Inc., 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pp. 770–778. IEEE Computer Society, 2016.

Yihui He and Song Han. ADC: automated deep compression and acceleration with reinforcement learning. *CoRR*, abs/1802.03494, 2018.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017*, pp. 1398–1406. IEEE Computer Society, 2017.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pp. 2261–2269. IEEE Computer Society, 2017.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (eds.), *Computer Vision - ECCV 2018 - 15th European Conference, Proceedings, Part XVI*, volume 11220 of *Lecture Notes in Computer Science*, pp. 317–334. Springer, 2018.

A Krizhevsky and G Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1, 01 2009.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2, [NIPS 1989*, pp. 598–605. Morgan Kaufmann, 1989.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Conference Track Proceedings*, 2017.

Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun. OICSR: out-in-channel sparsity regularization for compact deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, volume abs/1905.11664, 2019.

Chen Lin, Zhao Zhong, Wu Wei, and Junjie Yan. Synaptic strength for convolutional neural network. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pp. 10170–10179, 2018.

Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David S. Doermann. Towards optimal structured CNN pruning via generative adversarial learning. *CoRR*, abs/1903.09291, 2019.

Chuanjian Liu, Yunhe Wang, Kai Han, Chunjing Xu, and Chang Xu. Learning instance-wise sparsity for accelerating deep models. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pp. 3001–3007, 2019.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision, ICCV 2017*, pp. 2755–2763, 2017.

Jelena Luketina, Tapani Raiko, Mathias Berglund, and Klaus Greff. Scalable gradient-based tuning of continuous regularization hyperparameters. In Maria-Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 2952–2960. JMLR.org, 2016.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision, ICCV 2017*, pp. 5068–5076, 2017.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 2113–2122. JMLR.org, 2015.

Dushyant Mehta, Kwang In Kim, and Christian Theobalt. Implicit filter sparsification in convolutional neural networks. *CoRR*, abs/1905.04967, 2019.

Chaithanya Kumar Mummadi, Tim Genewein, Dan Zhang, Thomas Brox, and Volker Fischer. Group pruning using a bounded-lp norm for group gating and regularization. *CoRR*, abs/1908.03463, 2019.

Bo Peng, Wenming Tan, Zheyang Li, Shun Zhang, Di Xie, and Shiliang Pu. Extreme network compression via filter group approximation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (eds.), *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VIII*, volume 11212 of *Lecture Notes in Computer Science*, pp. 307–323. Springer, 2018.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4331–4340. PMLR, 2018.

Ricotti, Ragazzini, and Martinelli. Learning of word stress in a sub-optimal second order backpropagation neural network. In *IEEE 1988 International Conference on Neural Networks*, pp. 355–361 vol.1, July 1988. doi: 10.1109/ICNN.1988.23867.

Charbel Sakr and Naresh R. Shanbhag. Per-tensor fixed-point quantization of the back-propagation algorithm. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings*, 2015.

Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P. Namboodiri. Play and prune: Adaptive filter pruning for deep model compression. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pp. 3460–3466. ijcai.org, 2019.

Wenxiao Wang, Cong Fu, Jishun Guo, Deng Cai, and Xiaofei He. COP: customized deep model compression via regularized correlation-based filter-level pruning. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pp. 3785–3791. ijcai.org, 2019.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. NISP: pruning networks using neuron importance score propagation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*, pp. 9194–9203. IEEE Computer Society, 2018.

Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jin-Hui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pp. 883–894, 2018.

A Appendix

Example 1. *Deep Neural Network performs classification, where $\{(\boldsymbol{x}^n, \boldsymbol{y}^n)\}$ is sample datas. The activation function is differentiable, the derivatives of the optimal solution $\tilde{\boldsymbol{w}} = [\tilde{\boldsymbol{W}}^{l+}, \tilde{\boldsymbol{W}}^{l-}, \tilde{\boldsymbol{\mu}}]$ with respect to $\boldsymbol{\lambda}$ are:*

$$\nabla_{\boldsymbol{\lambda}} \boldsymbol{s}(\boldsymbol{\lambda})^* = -\nabla_{\tilde{\boldsymbol{w}}} g(\tilde{W_{i,j}^{l+}}, \tilde{W_{i,j}^{l-}}, \tilde{\mu}_t, \tilde{\mu}_{t+1}\lambda_1, \lambda_2)^{-1} \nabla_{\boldsymbol{\lambda}} g(\tilde{W_{i,j}^{l+}}, \tilde{W_{i,j}^{l-}}, \tilde{\mu}_t, \tilde{\mu}_{t+1}, \lambda_1, \lambda_2) \tag{18}$$

When the gradient of the weight parameter with respect to the hyperparameter is solved, the gradient of the hyperparameter can be given by the chain rule:$\nabla_{\boldsymbol{\lambda}} cv = \frac{\partial cv}{\partial \boldsymbol{s}(\boldsymbol{\lambda})^*} \nabla_{\boldsymbol{\lambda}} \boldsymbol{s}(\boldsymbol{\lambda})^*$. The following equation is just an example of four representative variables: $\tilde{W_{i,j}^{l+1+}}, \tilde{W_{i,j}^{l+1-}}, \tilde{\mu}_t, \tilde{\mu}_{t+1}$.

$$\begin{aligned} \nabla_{\boldsymbol{\lambda}} cv &= -\begin{pmatrix} -\beta \\ \beta \\ 0 \\ 0 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \alpha & -\alpha & 0 & 0 \\ -\alpha & \alpha & 0 & 0 \\ \mu_t & 0 & W_{i,j}^{l+1+} & 0 \\ 0 & \mu_{t+1} & 0 & W_{i,j}^{l+1-} \end{pmatrix}^{-1} \begin{pmatrix} 1 & W_{i,j}^{l+1} \\ 1 & -W_{i,j}^{l+1} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} && (19) \\ &= -\begin{pmatrix} -\beta \\ \beta \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \alpha & -\alpha \\ -\alpha & \alpha \end{pmatrix}^{-1} \begin{pmatrix} 1 & W_{i,j}^{l+1} \\ 1 & -W_{i,j}^{l+1} \end{pmatrix} && (20) \end{aligned}$$

$$\begin{aligned} \alpha &= \sum_{n=1}^{N} \sum_{k=1}^{m} \left\{ \frac{y_k^n (x_k^l)^2}{(o_k^n)^2} \left[\left(\frac{\partial o_k^n}{\partial a_k^{l+1}} \right)^2 - o_k^n \frac{\partial^2 o_k^n}{\partial (a_k^{l+1})^2} \right] \right\} + \lambda_2 && (21) \\ \beta &= \sum_{n=1}^{M} \sum_{k=1}^{m} \left(\tilde{y}_k^n \frac{1}{\tilde{o}_k^n} \right) \frac{\partial \tilde{o}_k^n}{\partial \tilde{a}_k^{l+1}} \tilde{x}_k^l && (22) \end{aligned}$$

$g \in \mathbb{R}^{1\times d}, \nabla_{\tilde{\boldsymbol{w}}} g \in \mathbb{R}^{d\times d}, \nabla_{\boldsymbol{\lambda}} g \in \mathbb{R}^{d\times 2}, \nabla_{\boldsymbol{\lambda}} \boldsymbol{s}(\boldsymbol{\lambda}) \in \mathbb{R}^{d\times 2}, \frac{\partial cv}{\partial \boldsymbol{s}(\boldsymbol{\lambda})} \in \mathbb{R}^{1\times d}, \nabla_{\boldsymbol{\lambda}} cv \in \mathbb{R}^{1\times 2}$. *Two elements in $\nabla_{\boldsymbol{\lambda}} cv$ are represented the cross-validation gradients of λ_1 and λ_2 separately.*