Collaborative Channel Pruning for Deep Networks

Hanyu Peng [1] Jiaxiang Wu [2] Shifeng Chen [1] Junzhou Huang [3]

Abstract

Deep networks have achieved impressive performance in various domains, but their applications are largely limited by the prohibitive computational overhead. In this paper, we propose a novel algorithm, namely collaborative channel pruning (CCP), to reduce the computational overhead with negligible performance degradation. The joint impact of pruned/preserved channels on the loss function is quantitatively analyzed, and such inter-channel dependency is exploited to determine which channels to be pruned. The channel selection problem is then reformulated as a constrained 0-1 quadratic optimization problem, and the Hessian matrix, which is essential in constructing the above optimization, can be efficiently approximated. Empirical evaluation on two benchmark data sets indicates that our proposed CCP algorithm achieves higher classification accuracy with similar computational complexity than other state-of-the-art channel pruning algorithms.

1. Introduction

In the past few years, deep learning models have been continuously boosting the state-of-the-art performance in various domains, ranging from image classification (He et al., 2016), segmentation (Chen et al., 2018), and object detection (Ren et al., 2015). However, such performance boost often comes at the cost of increasing the computational complexity and number of model parameters. This imposes a great challenge to efficient inference with such complicated models. Especially, for mobile devices and embedded systems, where the computation resources are rather limited, it is often infeasible to directly deploy deep networks due to the high latency and power consumption.

This work was done when Hanyu Peng was an intern at Tencent AI Lab. [1]Multimedia Laboratory, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences [2]Tencent AI Lab [3] Department of CSE, The University of Texas at Arlington. Correspondence to: Shifeng Chen <shifeng.chen@siat.ac.cn>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

There are a large collection of research papers focusing on accelerating and compressing deep networks to improve the inference efficiency. Among them, approaches based on channel pruning are proved to be quite effective since they have no extra requirements for the inference engine (He et al., 2017; Luo et al., 2017; Zhuang et al., 2018) and can be easily deployed. The basic idea is to reduce the number of input and/or output channels in convolutional layers without degrading the performance. Most approaches consider the contribution of each channel to the reconstruction (or discrimination) loss independently to determine which channels to be pruned, but the inter-channel relationship is rarely exploited.

In this paper, we investigate how the inter-channel relationship can be utilized to determine which channels should be pruned/preserved to minimize the performance degradation. We propose the collaborative channel pruning (CCP) algorithm to capture the dependency between channels in convolutional layers. For each convolutional layer, we quantitatively analyze the joint impact of pruned/preserved channels to the final loss function, based the second-order Taylor expansion. Afterwards, we reformulate the minimization of loss function as a linearly constrained 0-1 quadratic problem.

The linearly constrained 0-1 quadratic problems involves the Hessian matrix that is too huge to be efficiently computed or stored. To tackle this issue, we present an efficient approximation scheme for the Hessian matrix which is applicable to both regression and classification tasks. The approximated Hessian matrix can help construct the linearly constrained 0-1 quadratic programming problem without explicitly computing or storing the Hessian matrix. Since this optimization problem is NP-hard due to binary variables, we relax the binary constraint and obtain an approximate solution via sequential quadratic programming (SQP) (Boggs & Tolle, 1995). Finally, we select the top-k entries as preserved channels and prune the remaining channels. We further fine-tune the resulting compressed model to reduce its performance degradation.

To the best of our knowledge, collaborative channel pruning is the first to exploit the inter-channel dependency to determine the optimal combination of preserved channels. We demonstrate that our CCP algorithm outperforms state-of-the-art channel pruning approaches on CIFAR-10 (Krizhevsky, 2009) and ImageNet (Russakovsky et al., 2015) data sets. We summarize our contributions as follows:

- We quantitatively analyze the joint impact of pruned/preserved channels on the final loss function, and propose a novel channel pruning algorithm to exploit such inter-channel dependency information, which is rarely considered before.

- We formulate the channel selection process as a linearly constrained 0-1 quadratic optimization problem, and propose an efficient approach to tackle it.

- The proposed CCP algorithm can reduce FLOPs of ResNet-50 by 54.1% while the top-1 and top-5 accuracy on ImageNet is merely decreased by 0.83% and 0.33%, which outperforms all the state-of-the-art methods.

2. Related Work

Model compression has gained much attention in recent years. Various methods have been proposed to reduce the model size via network quantization (Rastegari et al., 2016) (Courbariaux et al., 2015) (Han et al., 2016), channel pruning (Ye et al., 2018), low-rank approximation (Denton et al., 2014). Network quantization methods quantize weights or activations by low-bit variables (Rastegari et al., 2016) or hash tables (Wu et al., 2016). Channel Pruning aims to remove channels with negligible performance degradation. (Han et al., 2015) proposed to remove small weights of pre-trained network and fine-tuned the compressed network by back propagation with regularization. Such a technique can significantly reduce the model size and guarantee competitive performance, but the unstructured sparsity may not enable efficient inference. (He et al., 2017) (Luo et al., 2017) proposed to preserve channels that approximate the outputs of pre-trained models. However, minimizing the constructing errors neglects the effect on loss function and may result in performance degradation. Low-rank approximation applies matrix (tensor) decomposition, such as Singular Value Decomposition (SVD) (Denton et al., 2014) to reduce parameters.

Many approaches to some criterion have been proposed for channel pruning, such as l_1-norm (Li et al., 2016) and l_2-norm (He et al., 2018a) of the filters. (Anwar et al., 2017) introduced structure pruning at various levels to save computational cost. (Alvarez & Salzmann, 2016) (Lebedev & Lempitsky, 2016) applied group sparsity regularizer to reduce redundant parameters. (Hu et al., 2016) proposed to remove unimportant filters based on their activations without affecting performance. (He et al., 2017) proposed a channel pruning scheme that approximately preserved the outputs of pre-trained network and reformulated the channel pruning problem as a LASSO problem. (Zhuang et al., 2018) proposed to use the discrimination-aware loss to detect filters that really contribute to the classification accuracy. (Luo et al., 2017) selected a subset of channels based on the statistic information computed from the next layer. (Yu et al., 2018) identified and detected important filter based on neural importance score. (Zhang et al., 2018) used parameter sharing approach based on the similarity between filters to dramatically reduce the model size with a slight loss of accuracy. (He et al., 2018b) proposed AutoML to determine the optimal pruning ratio of each layer, our work can act as the backbone channel pruning algorithm to produce a more accurate compressed model under the given pruning ratio. There is some theoretical analysis guarantees the error bound on the compressed network. (Arora et al., 2018) provided a generalization bound and analysed the compression is dependent to the noise stability properties of pre-trained networks.

(Molchanov et al., 2017) introduced a pruning method based on first-order Taylor expansion that interleaved greedy pruning with standard fine-tuning. However, the Hessian matrix was neglected for efficiency. (LeCun et al., 1990) and (Hassibi & Stork, 1993) both used second-order Taylor expansion to prune channels. The pruning criterion in (LeCun et al., 1990) assumed that the Hessian matrix was diagonal and the interaction between two different channels was omitted. In (Hassibi & Stork, 1993), the full Hessian matrix was used but weights were pruned in a one-by-one manner, instead of directly finding the optimal combination for pruning as in our method. Fisher pruning (Theis et al., 2018) also utilized the second-order Taylor expansion to approximate the cross entropy loss function and demonstrated superior performance combined with knowledge distillation. All previous works considered the channels independent regardless of the dependency between channels. In contrast, we formulate the inter-channel dependency with the Hessian matrix, of which each non-diagonal element corresponds to the interaction between two different channels. Our proposed method captures the inter-channel dependency among channels which is rarely exploited before.

3. Preliminaries

Given a training set $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$, where $\mathbf{x}_n$ denotes the n-th training sample's input features and $\mathbf{y}_n$ is the corresponding outputs. We aim to train a L-layer neural network to minimize the difference between desired and actual outputs under certain loss metrics. For the l-th convolutional layer, we denote its convolutional kernel as $\mathbf{W}^{(l)} \in \mathbb{R}^{k \times k \times c_i \times c_o}$, where k is the kernel size, c_i and c_o is the number of input and output channels, respectively. More specifically, we use $\mathbf{W}_i^{(l)} \in \mathbb{R}^{k \times k \times c_i}$ to denote the convolutional kernel associated with the i-th output channel.

Channel pruning seeks to minimize the loss function's value under sparsity constraints on convolutional kernels, which can be formulated as:

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}\left[f\left(\boldsymbol{\theta}; \mathbf{x}_n\right), \mathbf{y}_n\right] \\ &\text{s.t. } \left\|\boldsymbol{\beta}^{(l)}\right\|_0 = p_l, \forall l = 1, \ldots, L \end{aligned} \tag{1}$$

where the model parameter $\boldsymbol{\theta}$ consists of all the convolutional kernels $\{\mathbf{W}^{(l)}\}$ and binary-valued mask vectors $\{\boldsymbol{\beta}^{(l)}\}$. Each entry in the binary mask $\boldsymbol{\beta}^{(l)} \in \{0,1\}^{c_o}$ indicates whether the corresponding output channel should be pruned (0) or not (1), and p_l is the total number of preserved output channels. $f\left(\boldsymbol{\theta}; \mathbf{x}_n\right)$ is the network's outputs for the n-th training sample $\mathbf{x}_n$, and the choice of loss function $\mathcal{L}(\cdot)$ depends on the specific learning task. For instance, it can be cross-entropy loss for classification or least-square loss for regression.

4. Methods

In this section, we firstly analyze the impact of all the pruned (or preserved) channels on the final loss function with the second-order Taylor expansion. The resulting minimization problem involves the computation of Hessian matrix; therefore, we propose an efficient approximation of it. Finally, we transform the optimization into a constrained 0-1 quadratic programming problem, and demonstrate how it can be solved efficiently.

4.1. Impact of Pruned Channels

Let us consider channel pruning of a single convolutional layer, while keeping all the other layers fixed. For the i-th output channel, we reshape the corresponding convolutional kernel into a vector, denoted as $\mathbf{w}_i \in \mathbb{R}^{k^2 c_i}$, where the superscript (l) is dropped for simplicity. All the convolutional kernels can be either stacked into a 2-D matrix $\mathbf{W} \in \mathbb{R}^{c_o \times k^2 c_i}$ or concatenated into a 1-D vector $\mathbf{w} \in \mathbb{R}^{c_o k^2 c_i}$. The binary mask β_i indicates whether the i-th output channel should be pruned ($\beta_i = 0$) or not ($\beta_i = 1$).

With other layers' parameters treated as constants, the loss function can be written as a joint function of all the output channels' convolutional kernels and masks:

$$\mathcal{L}(\boldsymbol{\beta}, \mathbf{W}) \approx \mathcal{L}(\mathbf{W}) + \mathbf{g}^T \mathbf{v} + \frac{1}{2} \mathbf{v}^T \mathbf{H} \mathbf{v} \tag{2}$$

where $\mathbf{v} = \operatorname{vec}(\boldsymbol{\beta} \odot \mathbf{W} - \mathbf{W})$ is the flatten vector of $\boldsymbol{\beta} \odot \mathbf{W} - \mathbf{W}$ and $\odot$ denotes multiplication with broadcasting. We use $\mathbf{g}$ and $\mathbf{H}$ to denote the first and second-order derivatives:

$$\mathbf{g} = \nabla \mathcal{L}(\mathbf{w}), \ \mathbf{H} = \nabla^2 \mathcal{L}(\mathbf{w}) \tag{3}$$

We can further divide $\mathbf{g}$ and $\mathbf{H}$ into sub-vectors (one per output channel) and sub-matrices (one per output channel pair):

$$\mathbf{g} = \begin{bmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{c_o} \end{bmatrix}, \ \mathbf{H} = \begin{bmatrix} \mathbf{H}_{1,1} & \cdots & \mathbf{H}_{1,c_o} \\ \vdots & \ddots & \vdots \\ \mathbf{H}_{c_o,1} & \cdots & \mathbf{H}_{c_o,c_o} \end{bmatrix} \tag{4}$$

and thus the loss function can be rewritten as:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\beta}, \mathbf{W}) \approx\ & \mathcal{L}(\mathbf{W}) + \sum_{i=1}^{c_o} \mathbf{g}_i^T \left(\beta_i \mathbf{w}_i - \mathbf{w}_i\right) \\ & + \frac{1}{2} \sum_{i,j=1}^{c_o} \left(\beta_i \mathbf{w}_i - \mathbf{w}_i\right)^T \mathbf{H}_{ij} \left(\beta_i \mathbf{w}_i - \mathbf{w}_i\right) \end{aligned} \tag{5}$$

Assume a pre-trained model is provided in advance and its convolutional kernels $\{\bar{\mathbf{w}}_\mathbf{i}\}$ are kept unchanged during channel pruning. Since all the channels are not yet pruned, we have $\beta_i = 1, \forall i$. The task of channel pruning is find the optimal combination of $\{\beta_i\}$ that minimizes the loss function under the sparsity constraint:

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{\beta}, \bar{\mathbf{W}}\right) \approx\ & \mathcal{L}\left(\bar{\mathbf{W}}\right) + \sum_{i=1}^{c_o} \left(\beta_i - 1\right) \bar{\mathbf{g}}_i^T \bar{\mathbf{w}}_i \\ & + \frac{1}{2} \sum_{i,j=1}^{c_o} \left(\beta_i - 1\right)\left(\beta_j - 1\right) \bar{\mathbf{w}}_i^T \bar{\mathbf{H}}_{ij} \bar{\mathbf{w}}_j \end{aligned} \tag{6}$$

where both the first and second-order derivatives are computed based on the pre-trained model parameters. With the pre-trained model given, we can compute the following items in advance:

$$\begin{aligned} u_i &= \bar{\mathbf{g}}_i^T \bar{\mathbf{w}}_i \qquad , \forall i \\ s_{ij} &= \frac{1}{2} \bar{\mathbf{w}}_i^T \bar{\mathbf{H}}_{ij} \bar{\mathbf{w}}_j, \ \forall i, j \end{aligned} \tag{7}$$

which are independent to the choice of binary masks.

By removing constant items in (6), we arrive at the following sparsity-constrained optimization problem:

$$\begin{aligned} &\min \sum_{i=1}^{c_o} u_i \left(\beta_i - 1\right) + \sum_{i,j=1}^{c_o} s_{ij} \left(\beta_i - 1\right)\left(\beta_j - 1\right) \\ &\text{s.t. } \|\boldsymbol{\beta}\|_0 = p, \ \beta_i \in \{0,1\}, \ \forall i \end{aligned} \tag{8}$$

where p is the desired number of preserved output channels. In other words, for any pre-trained models, we can determine which channels to be pruned and which channels to be preserved by solving the above optimization, so that the loss function can be approximately minimized.

However, although the gradient $\bar{\mathbf{g}}$ can be computed in the linear time, the Hessian matrix $\bar{\mathbf{H}}$ is much harder to compute. Each Hessian sub-matrix is of size $k^2c_o \times k^2c_o$, which is quadratic to the number of parameters for one output channel. Furthermore, even if these Hessian matrices can be computed, how to store them is still challenging. Hence, we need to find an efficient approach to compute and store these Hessian matrices.

4.2. Approximated Hessian Matrix

4.2.1. Least-square Loss for Regression

Here, we firstly consider the least-square loss function for regression tasks. Formally, the loss function is defined as:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\|f_n(\mathbf{w}) - \mathbf{y}_n\|_2^2 \quad (9)$$

where $f_n(\mathbf{w}) = f(\mathbf{w}; \mathbf{x}_n)$ is the network's q-dimensional output vector for the n-th training sample $\mathbf{x}_n$, and $\mathbf{y}_n$ denotes the corresponding ground-truth label vector.

It is easy to see that the first-order derivative of least-square loss function is given by:

$$\nabla\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\nabla^T f_n(\mathbf{w})\left[f_n(\mathbf{w}) - \mathbf{y}_n\right] \quad (10)$$

where $\nabla f_n(\mathbf{w})$ is a $p \times d$ matrix and $d = c_o k^2 c_i$ is the dimension of $\mathbf{w}$. Each row in $\nabla f_n(\mathbf{w})$ consists of first-order derivatives related to a single output node.

The second-order derivative is given by:

$$\begin{aligned}\nabla^2\mathcal{L}(\mathbf{w}) &= \frac{1}{N}\sum_{n=1}^{N}\nabla^T f_n(\mathbf{w})\nabla f_n(\mathbf{w}) \\ &+ \frac{1}{N}\sum_{n=1}^{N}\sum_{p'=1}^{p}\nabla^2 f_{n,p'}(\mathbf{w})\left[f_{n,p'}(\mathbf{w}) - y_{n,p'}\right]\end{aligned} \quad (11)$$

where $f_{n,p'}(\mathbf{w})$ denotes the p'-th entry in the output vector. For a pre-trained model, the difference between $f_{n,p'}(\boldsymbol{\theta})$ and $y_{n,p'}$ should be quite small. Therefore, we omit the second term and the Hessian matrix can be approximated via:

$$\nabla^2\mathcal{L}(\mathbf{w}) \approx \frac{1}{N}\sum_{n=1}^{N}\nabla^T f_n(\mathbf{w})\nabla f_n(\mathbf{w}) \quad (12)$$

which only involves the first-order derivative. To simplify the upcoming description, we split the first-order derivative matrix into c_o sub-matrices, one per output channel:

$$\nabla f_n(\mathbf{w}) = \begin{bmatrix}\nabla_1 f_n(\mathbf{w}) \\ \vdots \\ \nabla_{c_o} f_n(\mathbf{w})\end{bmatrix} \quad (13)$$

where each sub-matrix is of size $p \times k^2 c_i$. The approximation of Hessian matrix can thus be rewritten as:

$$\nabla^2\mathcal{L}(\mathbf{w}) \approx \frac{1}{N}\sum_{n=1}^{N}\begin{bmatrix}\nabla_{n,1}^T\nabla_{n,1} & \cdots & \nabla_{n,1}^T\nabla_{n,c_o} \\ \vdots & \ddots & \vdots \\ \nabla_{n,c_o}^T\nabla_{n,1} & \cdots & \nabla_{n,c_o}^T\nabla_{n,c_o}\end{bmatrix} \quad (14)$$

where $\nabla_{n,i}$ is the short-hand for $\nabla_i f_n(\mathbf{w})$.

Consider its (i,j)-th sub-matrix $\nabla_{ij}^2\mathcal{L}(\mathbf{w})$, which accounts for the correlation between the i-th and j-th output channels' parameters. We discover that s_{ij}, which is needed in the optimization problem (8), can be efficiently obtained without explicitly computing this Hessian sub-matrix. More specifically, with model parameters $\bar{\mathbf{w}}$ given, we have:

$$\begin{aligned}s_{ij} &= \frac{1}{2}\bar{\mathbf{w}}_i^T \cdot \nabla_{ij}^2\mathcal{L}(\bar{\mathbf{w}}) \cdot \bar{\mathbf{w}}_j \\ &\approx \frac{1}{2}\bar{\mathbf{w}}_i^T \cdot \frac{1}{N}\sum_{n=1}^{N}\nabla_i^T f_n(\bar{\mathbf{w}})\nabla_j f_n(\bar{\mathbf{w}}) \cdot \bar{\mathbf{w}}_j \\ &= \frac{1}{2N}\sum_{n=1}^{N}\left[\nabla_i f_n(\bar{\mathbf{w}})\bar{\mathbf{w}}_i\right]^T\nabla_j f_n(\bar{\mathbf{w}})\bar{\mathbf{w}}_j \\ &= \frac{1}{2N}\sum_{n=1}^{N}\mathbf{v}_{n,i}^T(\bar{\mathbf{w}})\mathbf{v}_{n,j}(\bar{\mathbf{w}})\end{aligned} \quad (15)$$

where $\mathbf{v}_{n,i}(\bar{\mathbf{w}}) = \nabla_i f_n(\bar{\mathbf{w}})\bar{\mathbf{w}}_i \in \mathbb{R}^p$ can be computed with a time complexity of $\mathcal{O}\left(pk^2c_i\right)$. Therefore, the overall time complexity for s_{ij} is merely $\mathcal{O}\left(np + npk^2c_i\right)$, which is no longer quadratic to the number of model parameters.

4.2.2. Cross-entropy Loss for Classification

For the cross-entropy loss function, which is commonly used in classification problems, we can also approximately compute its Hessian matrix. The cross-entropy loss function is defined as:

$$\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{p'=1}^{p} y_{n,p'}\log f_{n,p'}(\mathbf{w}) \quad (16)$$

Similarly, we derive its first and second-order derivatives:

$$\begin{aligned}\nabla \mathcal{L}(\mathbf{w}) &= -\frac{1}{N}\sum_{n=1}^{N}\sum_{p'=1}^{p}\frac{y_{n,p'}}{f_{n,p'}(\mathbf{w})}\cdot\nabla f_{n,p'}(\mathbf{w}) \\ \nabla^2 \mathcal{L}(\mathbf{w}) &= -\frac{1}{N}\sum_{n=1}^{N}\sum_{p'=1}^{p}\Big[\frac{y_{n,p'}}{f_{n,p'}(\mathbf{w})}\cdot\nabla^2 f_{n,p'}(\mathbf{w}) \\ &\quad -\frac{y_{n,p'}}{f^2_{n,p'}(\mathbf{w})}\nabla f_{n,p'}(\mathbf{w})\nabla^T f_{n,p'}(\mathbf{w})\Big]\end{aligned} \tag{17}$$

To simplify the above equations, we introduce $\mathbf{z}_n \in \mathbb{R}^p$ where $z_{n,p'} = y_{n,p'}/f_{n,p'}(\mathbf{w})$ and diagonal matrix $\mathbf{\Sigma}_n \in \mathbb{R}^{p\times p}$ where $\Sigma_{n,p',p'} = y_{n,p'}/f^2_{n,p'}(\mathbf{w})$. Then, we can rewrite (17) as:

$$\begin{aligned}\nabla \mathcal{L}(\mathbf{w}) &= -\frac{1}{N}\sum_{n=1}^{N}\nabla^T f_n(\mathbf{w})\,\mathbf{z}_n \\ \nabla^2 \mathcal{L}(\mathbf{w}) &= \frac{1}{N}\sum_{n=1}^{N}\nabla^T f_n(\mathbf{w})\,\mathbf{\Sigma}_n\nabla f_n(\mathbf{w}) \\ &\quad -\frac{1}{N}\sum_{n=1}^{N}\sum_{p'=1}^{p}\frac{y_{n,p'}}{f_{n,p'}(\mathbf{w})}\cdot\nabla^2 f_{n,p'}(\mathbf{w})\end{aligned} \tag{18}$$

where $\nabla f_n(\mathbf{w}) \in \mathbb{R}^{p\times d}$. Here, we omit the second-order term $\nabla^2 f_{n,p'}(\mathbf{w})$ and approximate the Hessian matrix via:

$$\nabla^2 \mathcal{L}(\mathbf{w}) \approx \frac{1}{N}\sum_{n=1}^{N}\nabla^T f_n(\mathbf{w})\,\mathbf{\Sigma}_n\nabla f_n(\mathbf{w}) \tag{19}$$

It can be easily verified that with the above approximation, the pairwise correlation term s_{ij} in the optimization (8) can be efficiently computed. The time complexity is also $\mathcal{O}\left(np + npk^2c_i\right)$, same as that for the least-square loss.

4.3. Constrained 0-1 Quadratic Programming

Recall that to determine which channels to be pruned, we need to solve the optimization problem:

$$\begin{aligned}\min \; & \sum_{i=1}^{c_o} u_i(\beta_i - 1) + \sum_{i,j=1}^{c_o} s_{ij}(\beta_i - 1)(\beta_j - 1) \\ \text{s.t. } & \|\boldsymbol{\beta}\|_0 = p,\; \beta_i \in \{0,1\},\; \forall i\end{aligned} \tag{20}$$

By exploiting the 0-1 constraint imposed on $\boldsymbol{\beta}$, we can discard linear terms in the objective function by introducing the extended pairwise correlation matrix $\hat{\mathbf{S}} = [\hat{s}_{ij}]$ where:

$$\hat{s}_{ij} = \begin{cases} s_{ij}, & \text{if } i \neq j \\ s_{ij} + u_i - 2\sum_{j'=1}^{c_o} s_{ij'}, & \text{otherwise}\end{cases} \tag{21}$$

where the last term is based on the symmetric property of approximated Hessian matrix. Therefore, the optimization can be re-formulated as:

$$\begin{aligned}\min \; & \boldsymbol{\beta}^T\hat{\mathbf{S}}\boldsymbol{\beta} \\ \text{s.t. } & \mathbf{1}^T\boldsymbol{\beta} = p,\; \boldsymbol{\beta} \in \{0,1\}^{c_o}\end{aligned} \tag{22}$$

where $\mathbf{1}$ is a p-dimensional vector with all-one entries. This linearly-constrained 0-1 quadratic programming problem is NP-hard and intractable to solve. To obtain an approximate solution, we firstly drop the 0-1 constraint and only require that each entry of $\boldsymbol{\beta}$ lies in the interval $[0,1]$. The relaxed optimization is given by:

$$\begin{aligned}\min \; & \boldsymbol{\beta}^T\hat{\mathbf{S}}\boldsymbol{\beta} \\ \text{s.t. } & \mathbf{1}^T\boldsymbol{\beta} = p,\; \boldsymbol{\beta} \in [0,1]^{c_o}\end{aligned} \tag{23}$$

and can be solved by any linearly-constrained quadratic optimization algorithms. Here, we adopt the sequential quadratic programming (SQP) method proposed in (Boggs & Tolle, 1995) to tackle this problem. Since $\boldsymbol{\beta}$ is merely a c_o-dimensional vector where c_o is the number of output channels, the optimization process of SQP can be carried out very efficiently.

After obtaining a real-valued solution via SQP, we need to convert it into its 0-1 valued counterpart. We simply select the top-p entries with largest values and set them to ones and all the remaining ones are set to zeros.

A Graph Perspective. The above constrained 0-1 quadratic programming problem (22) can be interpreted from a graph perspective. In Figure 1, we visualize such a problem that $p = 4$ channels need to be preserved from $c_o = 6$ candidate channels. Each node represents a candidate output channel, and the edge connecting two nodes (including the self-to-self connection) is assigned with the corresponding weight $\hat{s}_{ij}$. Therefore, solving the optimization is equivalent to finding out a sub-graph of $p = 4$ nodes, so that the sum of edge weights in the sub-graph is minimized.

4.4. Summary

So far, we have presented our collaborative channel pruning algorithm. We briefly summarize the major workflow of proposed approach in Algorithm 1.

To start with, we initialize L groups of statistics ($\{u_i\}$ and $\{s_{ij}\}$), one per convolutional layer. Then, we traverse all the training samples, compute each sample's outputs and corresponding gradients, and incrementally update statistics for each layer. Afterwards, we take a layer-by-layer approach, from shallow layers to deeper ones, to determine which channels to be pruned. For each layer, we construct the pairwise correlation matrix from previously collected statistics, and obtain the binary mask by solving the corresponding optimization problem. Finally, after all layers

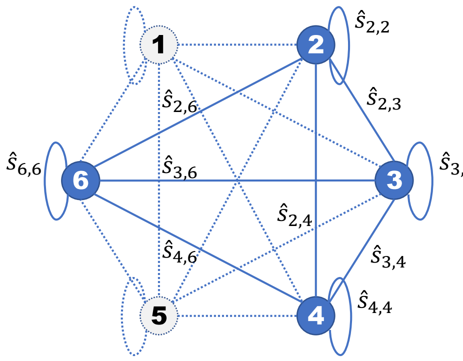

Figure 1. The visualization of a constrained 0-1 quadratic programming problem. Nodes with dashed outlines (#1 and #5) are pruned, and their corresponding edges are also removed and marked with dashed lines. Note that since the pairwise correlation matrix $\hat{\mathbf{S}}$ is symmetric, we only annotate one weight for each edge (*e.g.* for the edge annotated with $\hat{s}_{2,3}$, there is another weight $\hat{s}_{3,2}$ associated with it but not shown in the figure.

Algorithm 1 Collaborative Channel Pruning

```
Input: Training set $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$
Input: Pre-trained network $\boldsymbol{\theta}_0 = \{\mathbf{W}_0^{(l)}\}_{l=1}^L$
Output: Channel pruned network $\boldsymbol{\theta} = \{(\boldsymbol{\beta}^{(l)}, \mathbf{W}^{(l)})\}_{l=1}^L$
 1: initialize $\{u_i\}$ and $\{s_{ij}\}$ for all layers
 2: for $n = 1, \dots, N$ do
 3:     compute outputs and gradients for $(\mathbf{x}_n, \mathbf{y}_n)$
 4:     update $\{u_i\}$ and $\{s_{ij}\}$ for all layers
 5: end for
 6: for $l = 1, \dots, L$ do
 7:     compute pairwise correlation matrix $\hat{\mathbf{S}}$
 8:     solve (22) to obtain binary mask $\boldsymbol{\beta}^{(l)}$
 9: end for
10: fine-tune the model with binary masks $\{\boldsymbol{\beta}^{(l)}\}$
```

have been pruned, we fine-tune the model with preserved channels only to boost its prediction accuracy.

5. Experiments

In this section, we first evaluate our proposed collaborative channel pruning approach on two benchmark data sets, CIFAR-10 (Krizhevsky, 2009) and ILSVRC-12 (Russakovsky et al., 2015), and demonstrate its advantage over other channel pruning algorithms. Afterwards, we analyze the impact of different hyper-parameter settings on classification accuracy of the compressed model. Finally, we visualize the convergence behavior of solving the constrained 0-1 quadratic optimization problem via the SQP algorithm.

Specifically, we compare our approach against the following channel pruning algorithms:

- Channel Pruning (He et al., 2017)
- Pruning Filters (Li et al., 2016)
- ThiNet (Luo et al., 2017)
- Soft Pruning (He et al., 2018a)
- DCP - Discrimination-aware Channel Pruning (Zhuang et al., 2018)
- Neural Importance (Yu et al., 2018)
- AMC (He et al., 2018b)

5.1. Implementation Details

We implement our collaborative channel pruning method using PyTorch. Here, we attempt to prune ResNet (He et al., 2016) models since they are more accurate and efficient than the traditional AlexNet (Krizhevsky et al., 2012) and VGG (Simonyan & Zisserman, 2014) models. Therefore, it is more challenging and meaningful to compress such models. For the ILSVRC-12 data set, we prune ResNet-50 and use the pre-trained model provided by PyTorch as the uncompressed baseline model[1]. For the CIFAR-10 data set, we prune ResNet-56 and directly train an uncompressed baseline model from scratch. We adopt the standard data argumentation strategy as used in PyTorch's official examples. The extra computational cost for channel pruning before fine tuning is relatively small. For ResNet-56 on CIFAR-10, the time consuming is about 6 minutes, for ResNet-50 on ILSVRC-12, the computational time is about 26 minutes. These results are obtained on a Nvidia P40 GPU.

In the discrimination-aware channel pruning (Zhuang et al., 2018) algorithm, an extra group of classification losses are inserted into intermediate layers to improve their discriminative power. According to their results, this leads to the consistent improvement in the classification accuracy. In order to provide a fair comparison, we introduce a variant of our CCP algorithm, where an auxiliary classifier is inserted into the intermediate layer[2]. When fine-tuning the channel-pruned network, we use both the final classification loss and the newly added auxiliary classifier loss as supervision. The joint loss function is given by:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{final}(\boldsymbol{\theta}) + \lambda \mathcal{L}_{aux}(\boldsymbol{\theta}) \tag{24}$$

where $\mathcal{L}_{final}(\cdot)$ and $\mathcal{L}_{aux}(\cdot)$ are the final and auxiliary classifiers' losses, respectively. λ is the trade-off coefficient, and we set $\lambda = 10^{-4}$ on CIFAR-10 and $\lambda = 10^{-6}$ on ILSVRC-12. We denote this variant as "CCP-AC", where "AC" stands for "auxiliary classifier". More details about auxiliary classifier can be found in supplementary material.

[1] https://pytorch.org/docs/stable/torchvision/models.html

[2] For ResNet-56 on CIFAR-10, we add the auxiliary classifier on the top of layer "group1_block8_sum" (https://bit.ly/2B9zGx1); for ResNet-50 on ILSVRC-12, we add it on the top of layer "res4f" (https://bit.ly/2RHOAov).

Table 1. Comparison on the classification accuracy drop (compared against the uncompressed baseline model) and reduction in FLOPs of various channels pruning algorithms on the CIFAR-10 data set.

Method	Baseline	Pruned	
	Acc.	Acc. ↓	FLOPs
Channel Pruning (He et al., 2017)	92.80%	1.00%	50.0%
AMC (He et al., 2018b)	92.80%	0.90%	50.0%
Pruning Filters (Li et al., 2016)	93.04%	-0.02%	27.6%
Soft Pruning (He et al., 2018a)	93.59%	0.24%	52.6%
DCP (Zhuang et al., 2018)	93.80%	0.31%	50.0%
DCP-Adapt (Zhuang et al., 2018)	93.80%	-0.01%	47.0%
CCP		0.04%	47.0%
CCP	93.50%	0.08%	52.6%
CCP-AC		-0.19%	47.0%

Table 2. Comparison on the top-1/5 classification accuracy drop (compared against the uncompressed baseline model) and reduction in FLOPs of various channels pruning algorithms on the ILSVRC-12 data set. "-" means the corresponding result is not reported.

Method	Baseline		Pruned		
	Top-1 Acc.	Top-5 Acc.	Top-1 Acc. ↓	Top-5 Acc. ↓	FLOPs
Channel Pruning (He et al., 2017)	-	92.20%	-	1.40%	50.0%
ThiNet (Luo et al., 2017)	72.88%	91.14%	1.87%	1.12%	55.6%
Soft Pruning (He et al., 2018a)	76.15%	92.87%	1.54%	0.81%	41.8%
DCP (Zhuang et al., 2018)	76.01%	92.93%	1.06%	0.61%	55.6%
Neural Importance (Yu et al., 2018)	-	-	0.89%	-	44.0%
CCP			0.65%	0.25%	48.8%
CCP	76.15%	92.87%	0.94%	0.45%	54.1%
CCP-AC			0.83%	0.33%	54.1%

5.2. Comparison on CIFAR-10

The CIFAR-10 data set consists of 50k training samples and 10k test samples, drawn from 10 categories. We utilize all the training samples to compute the pairwise correlation matrix $\hat{\mathbf{S}}$. The channel-pruned network is fine-tuned for 200 epochs using SGD with batch size of 128; we set weight decay to 0.0005 and momentum to 0.8. The learning rate starts from 0.1 and is divided by 10 at 60-th, 120-th, and 160-th epochs.

In Table 1, we compare the classification accuracy of compressed models trained with our algorithm and baseline methods. We evaluate on two different pruning ratios: 1) $r = 0.35$ which prunes 35% channels per layer and the overall FLOPs is reduced by 47.0% and 2) $r = 0.40$ which prunes 40% channels per layer and the overall FLOPs is reduced by 52.6%. We discover that with similar reduction in FLOPs, our approach suffers smaller accuracy drop than all the other baseline methods. Furthermore, with the auxiliary classifier inserted, the classification accuracy is even increased by 0.19% with nearly 2 times speed-up than the original uncompressed model. This indicates that with the inter-channel dependency information exploited, our approach can better identify which channels are more important and should be preserved.

5.3. Comparison on ILSVRC-12

The ILSVRC-12 data set consists of over 1.2M training samples and 50k validation samples from 1000 categories. Here, we only use a subset of 25k training samples to compute the pairwise correlation matrix $\hat{\mathbf{S}}$ and determine which channels to be pruned for speed-up. The compressed model is then fine-tuned for 100 epochs with an initial learning rate of 10^{-3} and divided by 10 every 30 epochs. We use a batch size of 128 and set weight decay to 0.0001 and momentum to 0.9.

In Table 2, we report the top-1/5 accuracy drop and reduction in FLOPs of various channel pruning algorithms. Similarly, we present results for pruning ratios $r = 0.35$ and $r = 0.40$, which leads to 48.8% and 54.1% reduction in FLOPs. It is observed that with similar reduction in FLOPs (54.1% vs. 55.6%), our CCP approach outperforms the state-of-the-art DCP algorithm by 0.16% in the top-5 accuracy. With the auxiliary classifier enabled, the performance improvement is further increased to 0.28%. This also verifies that exploiting the inter-channel dependency is beneficial for selecting which channels to be pruned.

5.4. Parameter Study

Auxiliary Classifier's Coefficient. We vary the auxiliary classifier's coefficient λ to understand its effect on the compressed model's accuracy. We choose its value from $\{0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$ and attempt to prune the ResNet-56 model on the CIFAR-10 data set with a pruning ratio of $r = 0.35$. In Figure 2, we plot the accuracy drop under different values of λ. We discover that the accuracy drop is consistently negative, meaning that the compressed model is even more accurate than the original uncompressed one. This also validates that the introduction of auxiliary classifier can indeed boost the classification accuracy of compressed models.

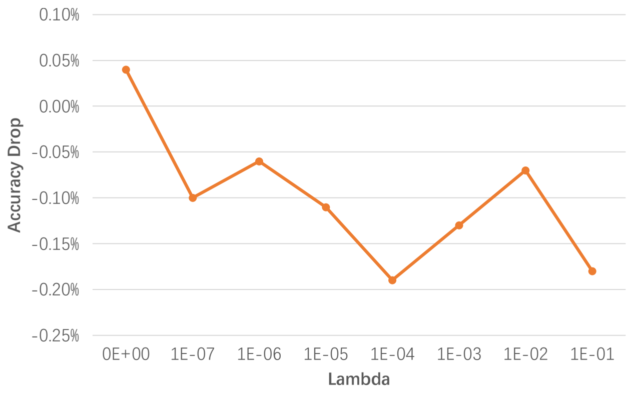

Figure 2. Comparison on the classification accuracy drop for ResNet-56 on the CIFAR-10 data set with varying auxiliary classifier's coefficient λ.

Pruning Ratio. We evaluate our approach for compressing the ResNet-56 model on the CIFAR-10 data set under different pruning ratios drawn from $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. In Figure 3, we report the accuracy drop under different pruning ratios. It is observed that with a relatively small pruning ratio, the compressed model even outperforms the uncompressed one, possibly due to the regularization effect introduced by channel pruning. As the pruning ratio increases, the accuracy drop becomes more significant, since with fewer channels in convolutional layers, the model's capacity is increasingly limited.

Convergence Behavior of SQP. Finally, we would like to verify whether the SQP algorithm we adopt can indeed solve the optimization problem (22) efficiently. We visualize a few convolutional layers' optimization process in Figure 4. We discover that the loss function converges within 10 iterations or even fewer, indicating that the SQP algorithm is indeed efficient in solving this optimization problem.

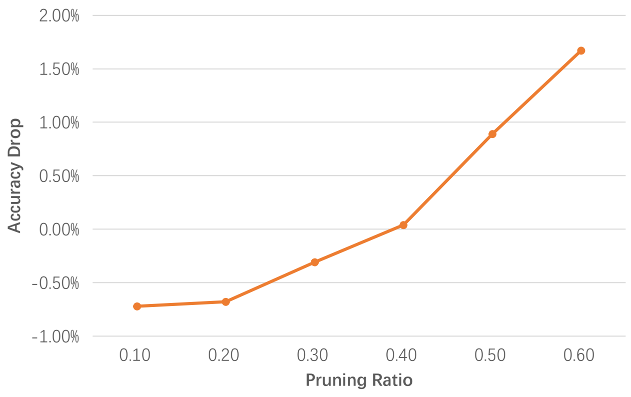

Figure 3. Comparison on the classification accuracy drop for ResNet-56 on the CIFAR-10 data set with varying pruning ratio.

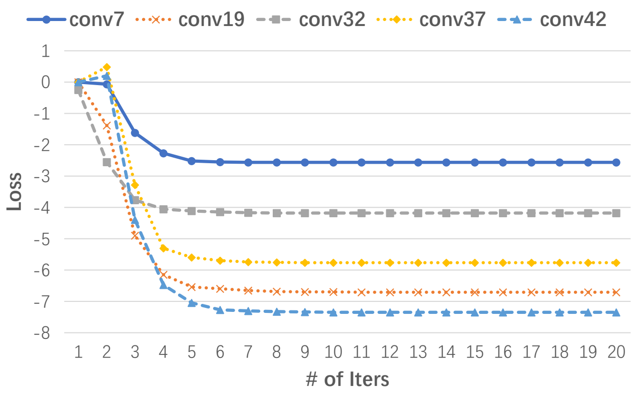

Figure 4. Comparison on the loss curve (loss function's value vs. # of iterations) of different convolutional layers.

6. Conclusion

In this paper, we present a novel collaborative channel pruning algorithm by exploiting the inter-channel dependency information, which is rarely considered before. The channel pruning process is then formulated as a constrained 0-1 quadratic optimization problem, and can be efficiently tackled with the sequential quadratic programming (SQP) solver. Extensive experiments validate the effectiveness of our proposed approach and demonstrate its advantage over other state-of-the-art channel pruning algorithms.

7. Acknowledgement

The work was funded by National Natural Science Foundation of China (U1713203) and Shenzhen Science and Technology Innovation Committee (Project KQJSCX20180330170238897)

References

Alvarez, J. M. and Salzmann, M. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2270–2278. Curran Associates, Inc., 2016.

Anwar, S., Hwang, K., and Sung, W. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32:1–32:18, Feb 2017.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. Stronger generalization bounds for deep nets via a compression approach. *CoRR*, abs/1802.05296, 2018.

Boggs, P. T. and Tolle, J. W. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(4):834–848, April 2018.

Courbariaux, M., Bengio, Y., and David, J.-P. BinaryConnect: training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3123–3131. Curran Associates, Inc., 2015.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1269–1277. Curran Associates, Inc., 2014.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1135–1143. Curran Associates, Inc., 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 164–171. Morgan-Kaufmann, 1993.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 1398–1406, Oct 2017.

He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. *CoRR*, abs/1808.06866, 2018a.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. AMC: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*, pp. 784–800, Sept 2018b.

Hu, H., Peng, R., Tai, Y., and Tang, C. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *CoRR*, abs/1607.03250, 2016.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105. Curran Associates, Inc., 2012.

Lebedev, V. and Lempitsky, V. Fast convnets using group-wise brain damage. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2554–2564, June 2016.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 598–605. Morgan-Kaufmann, 1990.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

Luo, J.-H., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 5068–5076, Oct 2017.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR)*, 2017.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. XNOR-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, pp. 525–542, 2016.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 91–99. Curran Associates, Inc., 2015.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Theis, L., Korshunova, I., Tejani, A., and Huszár, F. Faster gaze prediction with dense networks and fisher pruning. *CoRR*, abs/1801.05787, 2018.

Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. Quantized convolutional neural networks for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4820–4828, June 2016.

Ye, J., Lu, X., Lin, Z. L., and Wang, J. Z. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations (ICLR)*, 2018.

Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V. I., Han, X., Gao, M., Lin, C.-Y., and Davis, L. S. Nisp: Pruning networks using neuron importance score propagation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9194–9203, June 2018.

Zhang, D., Wang, H., Figueiredo, M., and Balzano, L. Learning to share: Simultaneous parameter tying and sparsification in deep learning. In *International Conference on Learning Representations (ICLR)*, 2018.

Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Cao, J., Wu, Q., Huang, J., and Zhu, J. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 883–894. Curran Associates, Inc., 2018.