A Main/Subsidiary Network Framework for Simplifying Binary Neural Networks

Yinghao Xu[1*] Xin Dong[2*] Yudian Li[4] Hao Su[3]

[1]Zhejiang University [2]Harvard University
[3]University of California, San Diego
[4]University of Electronic Science and Technology of China

[1]justimyhxu@zju.edu.cn [2]xindong@g.harvard.edu [4]daniellee2519@gmail.com [3]haosu@eng.ucsd.edu

Abstract

To reduce memory footprint and run-time latency, techniques such as neural network pruning and binarization have been explored separately. However, it is unclear how to combine the best of the two worlds to get extremely small and efficient models. In this paper, we, for the first time, define the filter-level pruning problem for binary neural networks, which cannot be solved by simply migrating existing structural pruning methods for full-precision models. A novel learning-based approach is proposed to prune filters in our main/subsidiary network framework, where the main network is responsible for learning representative features to optimize the prediction performance, and the subsidiary component works as a filter selector on the main network. To avoid gradient mismatch when training the subsidiary component, we propose a layer-wise and bottom-up scheme. We also provide the theoretical and experimental comparison between our learning-based and greedy rule-based methods. Finally, we empirically demonstrate the effectiveness of our approach applied on several binary models, including binarized NIN, VGG-11, and ResNet-18, on various image classification datasets. For binary ResNet-18 on ImageNet, we use only 78.6% filters, achieving better test error 49.87% ($= 50.02\% - 0.15\%$*) than the original model.*

1. Introduction

Deep neural networks (DNN), especially deep convolution neural networks (DCNN), have made remarkable strides during the last decade. From the first ImageNet Challenge winner network, AlexNet, to the more recent state-of-the-art, ResNet, we observe that DNNs are growing substantially deeper and more complex. These modern deep neural networks have millions of weights, rendering them both memory-intensive and computationally expensive. To reduce the computational cost, the research into network acceleration and compression emerges as an active field.

A family of popular compression methods are the DNN pruning algorithms, which are not only efficient in both memory and speed, but also enjoy relatively simple procedure and intuition. This line of research is motivated by the theoretical analysis and empirical discovery that redundancy does exist in both human brains and several deep models [8, 9]. We can categorize existing researches according to the level of the object, such as connection (weights)-level pruning, unit/channel/filter-level pruning, and layer-level pruning [30]. Connection-level pruning is the most widely studied approach, which produces sparse networks whose weights are stored as sparse tensors. Although both the *footprint memory* and the *I/O consumption* are reduced [14], such methods are often not helpful towards the goal of *computation acceleration* unless specifically-designed hardware is leveraged. This is because the dimensions of the weight tensor remain unchanged, though many entries are zeroed-out. As a well-known fact, the MAC operations on random structured sparse matrices are generally not too much faster than the dense ones of the same dimension. In contrast, *structural pruning techniques* [30], such as unit/channel/filter-level pruning, are more hardware-friendly, since they aim to produce tensors of reduced dimensions or having specific structures. Using these techniques, it is possible to achieve both computation acceleration and memory compression on general hardware and is common for deep learning frameworks.

We consider the structural network pruning problem for a specific family of neural networks – binary neural networks. A binary neural network is a compressed network of a general deep neural network through the *quantization strategy*. Convolution operations in DCNN[1] inherently involve matrix multiplication and accumulation (MAC). MAC operations become much more energy efficient if we use low-precision (1 bit or more) fixed-point number to approximate weights and activation functions (i.e., to quantify neurons) [4]. To

[*] Equal Contribution

[1] Fully connected layers can be implemented as convolution. Therefore, in the rest of the paper, we mainly focus on convolutional layers.

the extreme extent, the MAC operation can even be degenerated to Boolean operations, if both weights and activation are binarized. Such binary networks have been reported to achieve ∼58x computation saving and ∼32x memory saving in practice. However, the binarization operation often introduces noises into DNNs [21], thus the representation capacity of DNNs will be impacted significantly, especially when we also binarize the activation function. Consequently, binary neural networks inevitably require larger model size (more parameters) to compensate for the loss of representation capacity.

Although Boolean operation in binary neural networks is already quite cheap, even smaller models are still highly desired for low-power embedded systems, like smart-phones and wearable devices in virtual reality applications. Even though quantization (e.g., binarization) has significantly reduced the redundancy of each weight/neuron representation, our experiment shows that there is still heavy redundancy in binary neural networks, in terms of network topology. In fact, quantization and pruning are orthogonal strategies to compress neural networks: Quantization decrease the precision of parameters such as weights and activations, while pruning trims the connections in neural networks so as to attain the tightest network topology. However, previous studies on network pruning are all designed for full-precision models and cannot be directly applied for binary neural networks whose both weights and activations are 1-bit numbers. For example, it no longer makes any sense to prune filters by comparing the magnitude or L_1 norm of binary weights, and it is nonsensical to minimize the distance between two binary output tensors.

We, for the first time, define the problem of simplifying binary neural networks and try to learn extremely efficient deep learning models by combining pruning and quantization strategies. Our experimental results demonstrate that filters in binary neural networks are redundant and learning-based pruning filter selection is constantly better than those existing rule-based greedy pruning criteria (by weight magnitude or L_1 norm).

We propose a learning-based method to simplify binary neural network with a main-subsidiary framework, where the main network is responsible for learning representative features to optimize the prediction performance, while the subsidiary component works as a filter selector on the main network to optimize the efficiency. The contributions of this paper are summarized as follows:

- We propose a learning-based structural pruning method for binary neural networks to significantly reduce the number of filters/channels but still preserve the prediction performance on large-scale problems like the ImageNet Challenge.
- We show that our non-greedy learning-based method is superior to the classical rule-based methods in selecting which objects to prune. We design a main-subsidiary framework to iteratively learn and prune feature maps. Limitations of the rule-based methods and advantages of the learning-based methods are demonstrated by theoretical and experimental results. In addition, we also provide a mathematical analysis for L_1-norm based methods.
- To avoid gradient mismatch of the subsidiary component, we train this network in a layer-wise and bottom-up scheme. Experimentally, the iterative training scheme helps the main network to adopt the pruning of previous layers and find a better local optimal point.

2. Related Work

2.1. Pruning

Deep Neural Network pruning has been explored in many different ways for a long time. [15] proposed Optimal Brain Surgeon (OBS) to measure the weight importance using the second-order derivative information of loss function by Taylor expansion. [11] further adapts OBS for deep neural networks and has reduced the retraining time. Deep Compression [14] prunes connections based on weight magnitude and can achieve great compression ratio. The idea of dynamic masks [12] is also used for pruning. Other approaches used Bayesian methods and exploited the diversity of neurons to remove weights [25, 24]. However, these methods focus on pruning independent connection without considering group information. Even though they harvest sparse connections, it is still hard to attain the desired speedup on hardware.

To address the issues in connection-level pruning, researchers proposed to increase the group-sparsity by applying sparse constraints to the channels, filters, and even layers [30, 1, 27, 2]. [17] used LASSO constraints and reconstruction loss to guide network channel selection. [22] introduced L_1-Norm rank to prune filters, which reduces redundancy and preserves the relatively important filters using a greedy policy. [23] leverages a scaling factor from batch normalization to pruning channels. To encourage the scaling factor to be sparse, a regularization term is added to the loss function. On one hand, methods mentioned above are all designed for full-precision models and cannot be trivially transferred to binary networks. For example, to avoid introducing any non-Boolean operations, batch normalization in binary neural networks (like XNOR-Net) typically doesn't have scaling (γ) and shifting (β) parameters [4]. Since all weights and activation only have two possible values $\{1, -1\}$, it is also invalid to apply classical tricks such as ranking filters by their L_1-Norms, adding a LASSO constraint, or minimizing the reconstruction error between two binary vectors. On the other hand, greedy policies that ignore the correlations between filters cannot preserve all important filters.

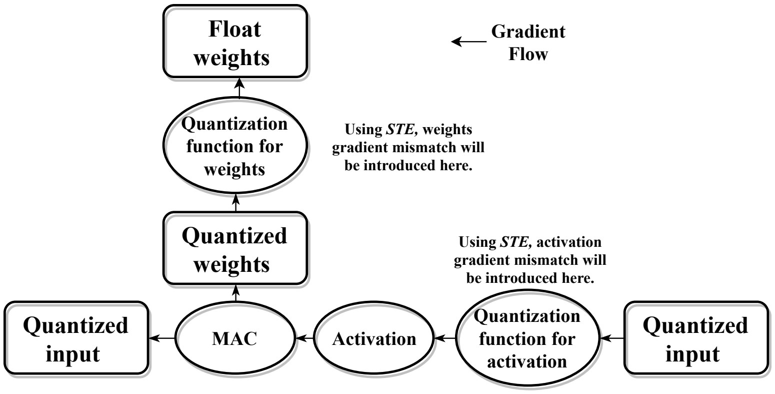
Float weights
Gradient Flow
Quantization function for weights
Using *STE*, weights gradient mismatch will be introduced here.
Quantized weights
Using *STE*, activation gradient mismatch will be introduced here.
Quantized input
MAC
Activation
Quantization function for activation
Quantized input

Figure 1. Gradient flow of binary neural networks during back-propagation. Rectangles represent the weight tensor and ellipses represent functional operation. In this paper, we use binary operation as a special quantization function. MAC is short for multiplication and accumulate operations, or the equivalent substitution like XNOR [4] in BNN.

2.2. Quantization

Recent work shows that full precision computation is not necessary for the training and inference of DNNs [13]. Weights quantization is thus widely investigated, e.g., to explore 16-bit [13] and 8-bit [10] fixed-point numbers. To achieve higher compression and acceleration ratio, extremely low-bit models like binary weights [6, 20] and ternary weights[32, 31, 29] have been studied, which can remove all the multiplication operations during computation. Weight quantization has a relatively milder gradient mismatch issue as analyzed in Section 3.1.2, and lots of methods can achieve comparable accuracy with full-precision counterparts on even large-scale tasks. However, the ultimate goal for quantization networks is to replace all MAC operations by Boolean operations, which naturally desires that both activation and weights are quantized, even binarized.

The activation function of quantized network has the form of a step function, which is discontinuous and non-differentiable. Gradient cannot flow through a quantized activation function during back-propagation. The *straight-through estimator* (STE) is widely adopted to circumvent this problem, approximating the gradient of step function as 1 in a certain range [18, 3]. [5] proposed the Half-wave Gaussian Quantization (HWGQ) to further reduce the mismatch between the forward quantized activation function and the backward ReLU [26]. Binary Neural Networks (BNN) proposed in [7] and [4] use only 1 bit for both activation functions and weights, ending up with an extremely smaller and faster network. BNNs inherit the drawback of acceleration via quantization strategy and their accuracy also needs further improving.

3. Approach

Let $F_b^i \in R^{N_i \times H_i \times W_i}$ denote binary input feature maps of the i-th layer in an I-layer binary neural network, where N_i, H_i, and W_i are the number of the input feature maps, height, and width of the activation map, respectively. Kernel weights $W_b^i \in R^{N_{i+1} \times N_i \times K_{i+1} \times K_{i+1}}$ in this layer are convolved with the input feature map F_b^i into output feature map F_b^{i+1}. Because both weights and activations are binary, we remove the subscripts of F_b and W_b for clarity. The goal of pruning is to remove certain filters $W_{n,:,:,:}^i$, $n \in \Omega$, where Ω is the indices of pruned filters. If a filter is removed, the corresponding output feature map of this layer (which is also the input feature map of next layer) will be removed, too. Furthermore, the input channels of all filters in the next layer would become unnecessary. If all filters in one layer can be removed, the filter-level pruning will upgrade to layer-level pruning naturally. The goal of our method is to remove as many filters as possible for binary neural networks which are already compact and have inferior numerical properties, thus this task is more challenging compared with pruning a full-precision model.

3.1. Subsidiary Component And main Networks

We borrow the ideas from binary network optimization to simplify binary networks. While it sounds tautological, note that the optimization techniques were originally invented to solve the quantization problem, but we will show that it can be crafted to solve the pruning problem for binary networks. A new binary network, called subsidiary component, acts as learnable masks to screen out redundant features in the main network, which is the network to complete classification tasks. Each update of the subsidiary component can be viewed as the exploration in the mask search space. We try to find a (local) optimal mask in that space with the help of the subsidiary component.

The process of training subsidiary and main networks is as follows:

3.1.1 Feature Learning – the Main Network

For layer i, the weights of subsidiary component $M^i \in R^{N_{i+1} \times N_i \times K_{i+1} \times K_{i+1}}$ are initialized by the uniform distri-

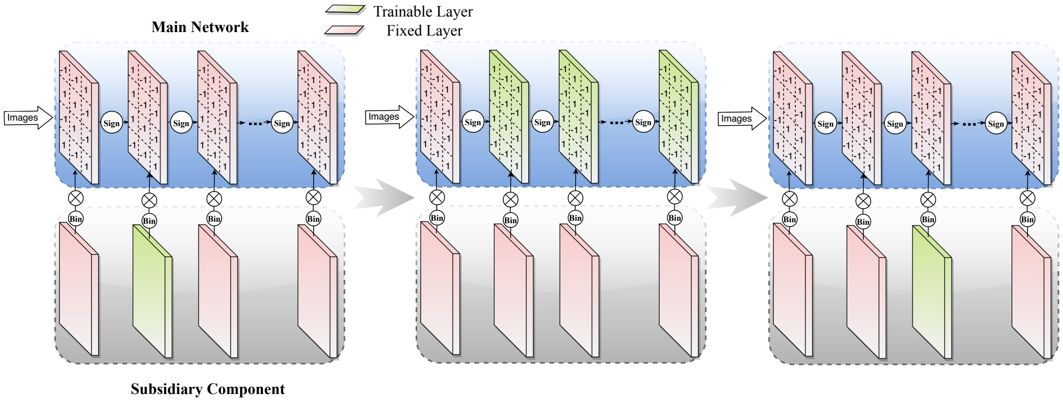

Figure 2. Pipline of our method. The main network in this figure is already pre-trained. From left to right: our subsidiary component training for i-th layer, main network retraining, and subsidiary component training for (i+1)-th layer.

bution: $M^i = U(-\sigma, \sigma)$. In practice, σ is chosen to be less than 10^{-5}. To achieve the goal of pruning filters, all elements whose first index is the same share the same value. $M_{n,o_1,p_1,q_1} = M_{n,o_2,p_2,q_2}, \quad \forall o, p, q$. Filter mask $O^i \in R^{N_{i+1} \times N_i \times K_{i+1} \times K_{i+1}}$ is an output tensor from the subsidiary component. In the first stage, we use the $Iden(\cdot)$ function (identity transformation) to get O^i.

$$O^i = Iden(M^i)$$

We apply the filter mask O^i to screen main network's weights W^i,

$$\hat{W}^i = O^i \otimes W^i$$

, where $\otimes$ is element-wise product. $\hat{W}^i$ denotes the weights of the main network after transformation, which is used to be convolved with the input feature maps, F^i, to produce the output feature maps F^{i+1}. Then, weights of the main network, $W^j, \ j \in [1, I]$, are set to be **trainable** while weights of the subsidiary component, $M^j, \ j \in [1, I]$, are **fixed**. Because subsidiary weights are fixed and initialized to be near-zero, it will not function in the *Feature Learning* stage, thus $\hat{W}^j \approx W^j, \ j \in [1, I]$. The whole main binary neural network will be trained from scratch.

3.1.2 Feature Selection – the subsidiary component

Training Subsidiary Component within a Single Layer i: After training the whole main network from scratch, we use a binary operator to select features in a layer-wise manner. In opposite to the previous *Feature Learning* stage, the weights of all layers $W^j, \ j \in [1, I]$ of the main network and the weights except layer i of the subsidiary component $M^j, \ j \in [1, I]/[i]$ are set to be **fixed**, while the subsidiary component's weights at the current layer M^i are **trainable** when selecting features for Layer i. The transformation function for the filter mask O^i is changed from $Iden(\cdot)$ to $Bin(\cdot)$ (*sign transformation + linear affine*),

$$O^i = Bin(M^i) = \frac{Sign(M^i) + 1}{2}$$

By doing this, we project the float-point M_i to binarized numbers ranging from 0 to 1. Elements in O_i which are equal to 0 indicate that the corresponding filters are removed and the elements of value 1 imply to keep this filter.

Since $Bin(\cdot)$ is not differentiable, we use the following function instead of the sign function in back propagation when training the subsidiary component M^i [18, 3],

$$f(x) = \begin{cases} x & -1 < x < 1 \\ 1 & x \geq 1 \\ -1 & x \leq -1 \end{cases} \tag{1}$$

Apart from the transformation, we also need to add regularization terms to prevent all O_i from degenerating to one, which is a trivial solution. So the loss function of training Layer i in the subsidiary component is,

$$\arg\min_{M^i} \ L_{cross\_entropy} + \alpha \cdot L_{reg} + \beta \cdot L_{distill} \tag{2}$$

$$L_{reg} = \|O_i\|_1$$

where $L_{cross\_entropy}$ is the loss on data and $L_{distill}$ is the distillation loss defined in (7).

Finally, we fix the layers $M^j, \ j \in [1, I]$ in the subsidiary component and layers before i in the main network (i.e., $W^j, \ j \in [1, i-1]$), and retrain the main layers after Layer i (i.e., $W^j, \ j \in [i, I]$).

Bottom-up Layer-wise Training for Multiple Layers: We showed how to train a layer in the subsidiary component above. To alleviate the gradient mismatch and keep away from the trivial solution during *Features Selection*, next, we propose a layer-wise and bottom-up training scheme for the subsidiary component: Layers closer to the input in the subsidiary component will be trained with priority. As Layer i is under training, all previous layers (which should have already been trained) will be fixed and subsequent layers will constantly be the initial near-zero value during training. There are three advantages of this training scheme.

First, as in (1), we use *STE* as in [18, 3] to approximate the gradient of the sign function. By *chain rule*, for each activation node j in Layer i, we would like to compute an "error term" $\delta_j^i = \frac{\partial L}{\partial a_j^i}$ which measures how much that node is responsible for any errors in the output. For binary neural networks, activation is also binarized by a sign function which need STE for back-propagation. The "Error term" for binary neural networks is given by,

$$\delta_j^i = Sign'(a_j^i)\cdot\sum_q w_{j,q}^{i+1}\delta_q^{i+1} \tag{3}$$

$$\frac{\partial L}{\partial M^i} = \frac{\partial L}{\partial O^i}\cdot\frac{\partial O^i}{\partial M^i} \tag{5}$$

$$\frac{\partial Sign(a_j^i)}{\partial a_j^i} = 1_{|a_j^i|\le 1} \tag{4}$$

$$\frac{\partial O^i}{\partial M^i} = \frac{1}{2}\cdot 1_{|\mathrm{M^i}|\le 1} \tag{6}$$

where (3) and (5) can be obtained by the *chain rule*, and (4) and (6) are estimated from *STE*, which will introduce gradient mismatch into back-propagation as shown in Figure 1. We refer (6) as weight gradient mismatch issue and (4) as activation gradient mismatch issue. They are two open problems in the optimization of binary neural networks, both caused by the quantization transform functions like $Sign(\cdot)$. Earlier layers (for both main and subsidiary networks) which are closer to the input have more serious gradient mismatch problem due to the chain rule. Starting from bottom layers, we can train and fix layers who are harder to train as early as possible for the subsidiary component. In addition, because of the retraining part in *Features Selection*, bottom-up training scheme allows bottom layers to be fixed earlier, as well. For a main network with K layers, the i-th layer will be retrained for i times in total. In other words, gradients for layers with the most serious gradient mismatch problem are the least frequently propagated, which effectively alleviates the propagation of error.

Second, the bottom-up layer-wise training scheme helps the main network to better accommodate the feature distribution shift caused by the pruning of previous layers. As mentioned before, the main difference in the motivation between our pruning method and rule-based methods is that we have more learnable parameters to fit the data by focusing on the final network output. With the bottom-up and layer-wise scheme, even if the output of Layer i changes, subsequent layers in the main network can accommodate this change by modifying their features.

Lastly and most importantly, we achieve higher pruning ratio by this scheme. According to our experiments, a straight-forward global training scheme leads to limited pruning ratio. Some layers are pruned excessively and hence damaged the accuracy, while some layers are barely pruned, which hurts the pruning ratio. The layer-wise scheme would enforce all layer to be out of the comfort zone and allow balancing between accuracy and pruning ratio.

3.1.3 Pipeline

The pipeline of our method is as follows:

1. Initialize weights of subsidiary component $M^j,\ j \in [1, I]$ with near-zero σ's.
2. Set $M^j,\ j \in [1, I]$ to be fixed, and train the whole main network from scratch.
3. Train starting from the first binary kernel. Each layer is the same as in the algorithm shown below:
    - Change the activation function for M^i from $Iden(\cdot)$ to $Bin(\cdot)$. And all other parameters apart from M^i are fixed. Train subsidiary component according to (2).
    - Fix the subsidiary layers $M^j,\ j \in [1, I]$ and main layers before i-th layer $W^j,\ j \in [1, i-1]$, and retrain main layers after i-th layer $W^j,\ j \in [i, I]$.

Algorithm 1 Algorithm of our pipeline

```
for $i = 1$ to $N$ do:
    Train Subsidiary component for Layer $i$ :
    $O^i \leftarrow Bin(M^i) = \frac{Sign(M^i)+1}{2}$
    $\hat{W}^i = O^i \otimes W^i$
    Fix $M^j, j \in [1, N]/[j]$, $M^i$ is learnable
    Fix $W^j, j \in [1, N]$
    $Loss = L_{cross\_entropy} + \alpha * L_{reg} + \beta * L_{distillation}$

    ReTrain Main Network:
    Fix $M^j, j \in [1, N]$
    Fix $W^j, j \in [1, i-1]$, $W^k$ is learnable, $k \in [i, N]$
    $Loss = L_{cross\_entropy}$
```

3.1.4 Distillation loss

Though pruning network filters is not an explicit transfer learning task, the aim is to guide the thin network to learn more similar output distributions with the original network. The model is supposed to learn a soft distribution but not a hard one as proposed in previous traditional classifier networks. Hence, we add a distillation loss to guide the training subsidiary component to be more stable, as shown in Figure 3.

$$L_{distill} = (p\|q) = H(p, q) - H(p) \tag{7}$$

We set p to be the original binary neural network distribution. Because the distribution is fixed, the $H(p)$ is a constant and can be removed from $L_{distill}$. It means that the distillation loss can be written as

$$L_{distill} = -\sum_{i=1}^{M} log\frac{exp(z_i/T)}{\sum_{j=1}^{M} exp(z_j/T)} \times \frac{exp(t_i/T)}{\sum_{j}^{M} exp(t_j/T)}$$

where z_i and t_i represent the final output of the pruned and original networks before the softmax layer. T is a temperature parameter for the distillation loss defined in [19]. We set T as 1 in practice. M is the number of classes.

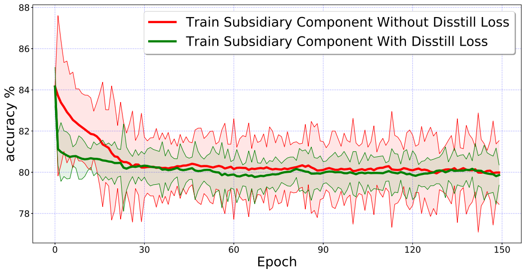

Figure 3. The learning curve for the Subsidiary Component. The red line refers to the learning curve without the distillation loss, and the red background represents the learning curve variance of every epoch. The green line and background represent the subsidiary component learning curve and the variance of each epoch. Clearly, the distillation loss makes the training procedure more stable.

3.2. Comparison with rule-based methods

Previous methods use rules to rank the importance of each filter and then remove the top k least important filters. The rules can be weight magnitude, e.g., measured by the L_1 norm, or some other well-designed criteria.

Studies in this line share the same motivation that individual filters have their own importance indication, and filters with less importance can be removed relatively safely. This assumption ignores interactions among filters. As mentioned before, rule-based pruning algorithms use a greedy way to prune filters, i.e., they assume that individual filters behave independently and their own importance (or function) for representation learning. We give a theoretical analysis in Section 3.3 about this point. In fact, pruning filters independently may cause problems when filter are strongly correlated. For example, if two filters have learned the same features (or concepts), these two filters may be pruned out together by rule-based methods, because their rankings are very close. Clearly, pruning one of them is a better choice.

However, almost all these criteria are based on value statistics and are completely unsuitable for the binary scenario with only two discrete values. One possible pruning method is to exhaustively search the optimal pruning set, but this is NP-Hard and prohibitive for modern DNNs that have thousands of filters. Our method uses the subsidiary component to "search" the optimal solution. Our soft "search" strategy is gradient-based and batch-based compared to exhaustive search, and it is much more efficient.

3.3. Relation to L_1-Norm pruning

If our main network is full-precision, the L_1-Norm based pruning technique would be strongly relevant to our method, except that we target at optimizing the final output of the network, whereas the L_1-Norm based method greedily controls the perturbation of the feature map in the next layer.

Suppose that $W = [w_1; \dots; w_n]$ is the original filters, and each row $w_i \in \mathbb{R}^{1\times m}$ is a vectored filter. $W' = [w'_1; \dots; w'_n]$ is the pruned filters. For some input $Z \in \mathbb{R}^{m\times n}$, each row of Z can be viewed as a patch sampled from an image. Let $\Delta w_i \equiv w_i - w'_i$. Then, the L_1-Norm approach minimizes the upper bound of the following problem: $\max_{\|x\|_\infty < \mathcal{T}} \|WX - W'X\|_2$. To see this, note

$$\|(W-W')X\|_2 = \| \begin{bmatrix} \Delta w_1 X \\ \dots \\ \Delta w_n X \end{bmatrix} \|_2 \le \sum_i \|\Delta w_i X\|_2 \tag{8}$$

$$\le \sum_i \|\Delta w_i X\|_1 \le \sum_i \|\Delta w_i\|_1 \|X\|_\infty \le \sum_i \|\Delta w_i\|_1 \mathcal{T} \tag{9}$$

where $\mathcal{T}$ is a constant. (1) and (2) are derived by Minkowski's Inequality and Hölder's Inequality. To minimize $\sum_i \|\Delta w_i\|_1$ by zeroing-out a single row w_i, obviously, the solution is to select the one with the smallest L_1-Norm.

However, note that this strategy cannot be trivially applied for binary networks, because the L_1-Norm for any filter that is a $\{-1, +1\}$ tensor of the same shape is always identical.

3.4. Relation to LASSO regression based least reconstruction error pruning

Previous work [16] uses the LASSO regression to minimize the reconstruction error of each layer: $min \ \|Y - \sum_{i=1}^{L} \beta_i X_i W_i\|_F^2, \ \ \|\beta\|_0 \le C'$. Solving this L_0 minimization problem is NP-hard, so the L_0 regularization is usually relaxed to L_1. In the binary/quantization scenario, activations only have two/several values and the least reconstruction error is not applicable. Instead of minimizing the reconstruction error of a layer, our method pays attention on the final network output with the help of the learnable subsidiary component. We directly optimize the discrete variables of masks (a.k.a subsidiary component) without the relaxation.

4. Experiments

To evaluate our method, we conduct several pruning experiments for VGG-11, Net-In-Net (NIN), and ResNet-18 on CIFAR-10 and ImageNet. Since our goal is to simplify binary neural networks, whose activation and weights are both 1-bit, all main models and training settings in our experiments inherit from XNOR-Net [28]. Since we are, to the best of our knowledge, the first work to define filter-level pruning for binary neural networks, we proposed a rule-based method by ourselves as the baseline. Instead of ranking filters according to the L_1-Norm [22], we use the magnitude of each filter's scaling factor (MSF) as our pruning criterion. Inspired by

[22], we test both the "prune once and retrain" scheme [2] and the "prune and retrain iteratively" scheme[3]. Apart from this, we have done another comparison that we random initialized the pruned network through our methods, and train a smaller binary neural network from scratch.

As pointed out in [28] we set weights of the first layer and last layer as full-precision, which also means that we only do pruning for the intermediate binary layers. We measure effectiveness of pruning methods in terms of PFR, the ratio of the number of pruned filters to original filter number, and error rate before and after retraining. For error ratio, smaller is better. For PFR, larger is better. For CIFAR-10, when training the main network, learning rate starts from 10^{-4}, and learning-rate-decay is equal to 0.1 for every 20 epochs. Learning rate is fixed with 10^{-3} when training the subsidiary component. For ImageNet, we set a constant learning rate of 10^{-3} for the subsidiary component and main work.

For fair comparison, we control PFR for each layer of these methods to be the same to observe the final Retrain-Error. In Figure **??**, MSF-Layerwise refers to the "prune once and retrain" scheme, and the MSF-Cascade refers the "prune and retrain iteratively" scheme. The first three figures of experiments were done on the CIFAR-10 dataset. The last figure refers to results on Imagenet.

4.1. NIN and VGG-11 On CIFAR-10

NIN is a fully convolutional network, using two 1×1 convolution layers instead of fully connected layer, and has quite compact architecture. VGG-11 is a high-capacity network for classification. VGG-11 on CIFAR-10 consists of 8 convolutional layers(including 7 binary layers) and 1 fully connected layer. Batch normalization is used between every binary convolution and activation layer, which makes the training process more stable and converges with high performance. For both MSF-Layerwise and MSF-Cascade, with the same PFR, the performance is worse than us. With $30\% \sim 40\%$ of pruning filter ratio, the pruned network error rate only increased $1\% \sim 2\%$.

4.1.1 Learning Rate is Important

An interesting phenomenon is observed when training subsidiary components for different models. We try different learning rates in our experiments and observe it impacts final convergent point a lot as shown in Figure 4. The relatively smaller learning rate (10^{-4}) will converge with lower accuracy and higher pruning number; however, the larger learning rate (10^{-3}) leads to the opposite result.

One possible explanation is that the solution space of the high-dimensional manifold for binary neural networks is more discrete compared to full-precision networks, so it is difficult for a subsidiary component to jump out of a locally optimal point to a better one. Moreover, in the binary scenario, larger learning rate will increase the frequency of value changing for weights. Our motivation is to use a learnable subsidiary components to approximate exhaustive search, so using a larger learning rate will enable the subsidiary component to "search" more aggressively. A large learning rate may be unsuitable for normal binary neural networks like the main network in this paper, but it is preferred by the subsidiary component.

4.1.2 Initialization of subsidiary component is NOT Sensitive

As mentioned in section 3.1.1, we use the uniform distribution to initialize the mask. According to the expectation of the uniform distribution, $E(PNR) = 0.5$, where PNR is the ratio of the number of positive elements in subsidiary weights to size of weights. However, since we use $Sign(\cdot)$, different PNR may impact the result a lot. We conduct six experiments on different models across different layers and show that initialization with 0.4, 0.6, 1.0 PNR will all converge to the same state. However, when PNR is 0.2, final performance will be very poor. A possible reason is that the number of filters thrown out by the initialization is too large, and due to the existence of the regularization term, the network's self-adjustment ability is limited and cannot converge to a good state. Hence we recommend the PNR to be intialized to greater than 0.4.

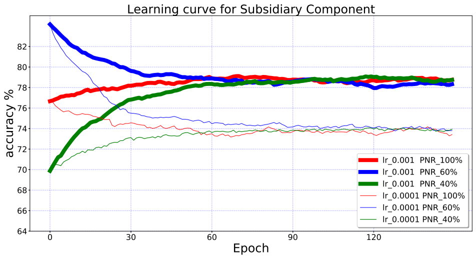

Figure 4. Learning curve for subsidiary component. We train the subsidiary component with different learning rate. These curves are smoothed for directly seeing the trend of the learning Subsidiary Component. All dotted lines represent the learning curve of the large learning rate 10^{-3}, the normal lines represent the learning curves of the small learning rate 10^{-4}.

4.2. ResNet on CIFAR-10 and ImageNet

Compared with NIN and VGG-11, ResNet has identity connections within residual block and much more layers. As the depth of network increases, the capacity of network also increases, which then leads to more redundancy. From experimental results, we find that when the identification mapping network has a downsampling layer, the overall sensitivity of the residual block will increase. Overall result for ResNet on

[2] Prune filters of multiple layers at once and retrain them until the original accuracy is restored

[3] Prune filters layer by layer and then retrain iteratively. The model is retrained before pruning the next layer for the weights to adapt to the changes from the pruning process.

Table 1. Overall results

Method	**Model**	**Original Error(%)**	**Retrain Error(%)**	**PFR(%)**
Smaller-Network	NIN	15.79%	40.27%	33.05%
MSF-Layerwise	NIN	15.79%	19.28%	33.05%
MSF-Cascade	NIN	15.79%	17.39%	33.05%
Our Method	NIN	15.79%	**16.89**%	33.05%
Smaller-Network	VGG-11	16.13%	18.01%	39.70%
MSF-Layerwise	VGG-11	16.13%	19.59%	39.70%
MSF-Cascade	VGG-11	16.13%	18.79%	39.70%
Our Method	VGG-11	16.13%	**18.03**%	39.70%
Smaller-Network	ResNet-18	12.11%	23.41%	39.89%
MSF-Layerwise	ResNet-18	12.11%	16.44%	39.89%
MSF-Cascade	ResNet-18	12.11%	14.63%	39.89%
Our Method	ResNet-18	12.11%	**13.61**%	39.89%
MSF-Layerwise	ResNet-18	50.02%	51.33 %	21.40%
MSF-Cascade	ResNet-18	50.02%	50.56%	21.40%
Our Method	ResNet-18	50.02%	**49.87**%	21.40%

Table 2. FLOPs and Memory usage for our pruned model

		FLOPs	Speedup	Memory Usage	Memory saving
ResNet-18	Our Pruned Model	1.46×10^8	12.39×	30.87Mbit	12.11×
	XNOR-NET	1.67×10^8	10.86×	33.70Mbit	11.10×
	Full-precision Res-Net	1.81×10^9	—	374.1Mbit	—-

CIFAR-10 is shown in table (1), and statistics for each layer can be found in Appendix.

We further verify our method with ResNet-18 on ImageNet. α can be set from 10^{-7} to 10^{-9} depending on the expected PFR, the accuracy and pruning ratio are balanced before retraining. After 20 epoches retraining for each layer, the final PFR is 21.4%, with the retrained error has decreased from 50.02% to 49.87%.

4.3. Efficiency and Memory Usage Analysis

In this section, we will analyze the speedup and memory saving of our pruned model and compare with XNOR-Net and full-precision network in ResNet-18.

The memory usage is the summation of number bits of all weights within one model. In addition, we use FLOPs to measure the efficiency for our pruned model. Because of the binary operation can implemented in XNOR operation and bit-counting in 64 parallel. So the final FLOPS are composed of the full-precision multiplication plus 1/64 1-bit multiplication.

We keep the first convolution layer and the last fully-connected layer to be real-valued and keep other weights and activations in the whole network are all binarized. As shown in Table 2, our pruned ResNet-18 model for ImageNet speed up $12.39\times$ and reduces $12.11\times$ memory usage in theory, but our test accuracy outperformed 0.15% compared with the full-precision ResNet-18 Model. We also achieves up to 14.10% higher speedup ratio and 9.09% memory saving ratio compared with XNOR-Net, saying that our model requires less memory and fewer FLOPs.

5. Conclusion

In this paper, we, for the first time, define the filter-level pruning problem for binary neural networks and propose a novel learning-based main/subsidiary network framework. Extensive experimental results on CIFAR and ImageNet demonstrate that the proposed main/subsidiary network framework and the novel training methods show efficiency for pruning of binary neural networks. What's more, our method is also friendly to pruning problem for quantized networks. In the future, we will explore more advanced learning algorithms for subsidiary part of the framework, because the learning-based framework for pruning has important value and will be treated as future work either.

References

[1] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

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

[3] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] A. Biswas and A. P. Chandrakasan. Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications. *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, 61:488–490, 2018.

[5] Z. Cai, X. He, J. Sun, and N. Vasconcelos. Deep Learning with Low Precision by Half-wave Gaussian Quantization. 2017.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. 2016.

[8] L. De Vivo, M. Bellesi, W. Marshall, E. A. Bushong, M. H. Ellisman, G. Tononi, and C. Cirelli. Ultrastructural evidence for synaptic scaling across the wake/sleep cycle. *Science*, 355(6324):507–510, 2017.

[9] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.

[10] T. Dettmers. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561*, 2015.

[11] X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[12] Y. Guo, A. Yao, and Y. Chen. Dynamic Network Surgery for Efficient DNNs. *Neural Information Processing Systems*, 2016.

[13] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[14] S. Han. Learning Both Weights and Connections for Efficient Neural Networks. *NIPS*, 346(8988):1500–1501, 2015.

[15] B. Hassibi, D. G. Stork, G. Wolff, and T. Watanabe. Optimal brain surgeon: Extensions and performance comparisons. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 263–270, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[16] Y. He, X. Zhang, and J. Sun. Channel Pruning for Accelerating Very Deep Neural Networks.

[17] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.

[18] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning. *Coursera, video lectures*, 264, 2012.

[19] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[20] Q. Hu, P. Wang, and J. Cheng. From hashing to cnns: Training binaryweight networks via hashing. *arXiv preprint arXiv:1802.02733*, 2018.

[21] H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein. Training Quantized Nets: A Deeper Understanding. *Advances in Neural Information Processing Systems*, 2017.

[22] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations*, 2016.

[23] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2755–2763. IEEE, 2017.

[24] Z. Mariet and S. Sra. Diversity networks. *Proceedings of ICLR*, 2016.

[25] D. Molchanov, A. Ashukha, and D. Vetrov. Variational Dropout Sparsifies Deep Neural Networks. *International Conference on Machine Learning*, 2017.

[26] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[27] A. Polyak and L. Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015.

[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[29] P. Wang and J. Cheng. Fixed-point factorized networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 3966–3974. IEEE, 2017.

[30] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning Structured Sparsity in Deep Neural Networks. *Advances in Neural Information Processing Systems*, page 10, 2016.

[31] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[32] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.