Latent Weight-based Pruning for Small Binary Neural Networks

Tianen Chen
University of Wisconsin–Madison
tianen.chen@wisc.edu

Noah Anderson
Stanford University
noah446@stanford.edu

Younghyun Kim
University of Wisconsin–Madison
younghyun.kim@wisc.edu

ABSTRACT

Binary neural networks (BNNs) substitute complex arithmetic operations with simple bit-wise operations. The binarized weights and activations in BNNs can drastically reduce memory requirement and energy consumption, making it attractive for edge ML applications with limited resources. However, the severe memory capacity and energy constraints of low-power edge devices call for further reduction of BNN models beyond binarization. Weight pruning is a proven solution for reducing the size of many neural network (NN) models, but the binary nature of BNN weights make it difficult to identify insignificant weights to remove.

In this paper, we present a pruning method based on latent weight with layer-level pruning sensitivity analysis which reduces the over-parameterization of BNNs, allowing for accuracy gains while drastically reducing the model size. Our method advocates for a heuristics that distinguishes weights by their latent weights, a real-valued vector used to compute the pseduogradient during backpropagation. It is tested using three different convolutional NNs on the MNIST, CIFAR-10, and Imagenette datasets with results indicating a 33%–46% reduction in operation count, with no accuracy loss, improving upon previous works in accuracy, model size, and total operation count.

KEYWORDS

binary neural network, pruning, latent weight

ACM Reference Format:
Tianen Chen, Noah Anderson, and Younghyun Kim. 2023. Latent Weight-based Pruning for Small Binary Neural Networks. In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3566097.3567873

1 INTRODUCTION

Despite the unprecedented success of machine learning (ML), bringing intelligence to resource-constrained edge devices has not seen similar success. While neural network (NN) models are rapidly growing in complexity and size to serve more and more sophisticated applications, the gap between their compute requirements and the capabilities of edge devices has only been widening. Specifically, for edge ML, the limited storage and memory capacity has been identified as a major hindrance [18]. The recent emergence of binary neural networks (BNNs) has shed some light on the possibility of making ML models smaller, in which all the weights and activations are binarized to either +1 or -1 [11] Binarized weights and activations require less memory and storage than their full-precision (floating-point or integer) counterparts, and they can also be processed with simple, low-power bit-wise logic units instead of complex, power-hungry arithmetic units. It makes BNNs highly suitable for ML applications on edge devices with small memory and storage, and thin energy budget.

However, despite the dramatic size reduction of binarization, BNN models still require further compression to be ported onto more severely resource-constrained platforms. Compression methods have been proposed to further optimize BNNs, including computation skipping [6], and bit-level data pruning [16]. Weight pruning is a widely applicable model compression technique that removes unnecessary or unimportant weights from the network [8]. In traditional full-precision NN models, unnecessary or unimportant weights can be easily identified by their small magnitude during the forward pass. Removing such near-zero weights has only a minimal impact on the output accuracy, and, in fact, it can even reach a "sweet spot" in the model where accuracy can surpass the original unpruned model accuracy due to the reduction of overfitting from overparameterization of the model, in addition to performance gains due to reduced computations [7]. This form of unstructured pruning comes at a potential hardware overhead cost identifying the sparsity within the weight matrix. However, compression methods can overcome potential hardware overhead by employing methods through quantization, encoding, and weight permutation [4, 7]. Some forms of extremely low-power networks, such as the combinational neural network, will require no hardware overhead when identifying sparsity within the matrix by simply removing the circuit component corresponding to the weight element [3, 19]. However, weight pruning for BNN models is not a straightforward problem since the magnitude of all weights is strictly 1, regardless of their sign, and thus magnitude cannot serve as an indicator of the weights' importance. Therefore, BNN pruning requires a new significance metric to replace the weight magnitude.

In this paper, we propose to use *latent weights* for pruning. Latent weights are real-valued weights that are used to obtain the pseudogradient vector during backpropagation as the real gradient vector cannot be obtained from binary weights [9]. We present a model compression technique that identifies the layer that has the greatest potential to improve the compression ratio at a time and prunes the layer based on the latent weights. The proposed technique includes an effective method to find the target layers based on the impact of pruning on the output accuracy without time-consuming model exploration. As a result, the proposed technique can achieve a dramatic reduction in model size and operation count, and reach the pareto-optimal of compressed networks that suffer no accuracy loss.

ASPDAC '23, January 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567873

The contributions of this paper are as follows:

- We present a latent weight-based pruning technique that selects layers that can be pruned with the minimum impact on the output accuracy and prune the layers based on latent weights.
- We introduce a multidimensional analysis of pruning layer-by-layer and include an optimization algorithm that intelligently minimizes a BNN that selectively prunes error-tolerant insensitive layers.
- We show experimental results that indicate a highly-optimized form of BNN pruning that decrease BOPs (binary operations) and model size by 46% and 27%, respectively, while incurring a small accuracy gain of +0.4% on the CIFAR-10 dataset, and similar results on other datasets. Our work is the first that can achieve a significant reduction in model size even without any accuracy loss.

2 BACKGROUND AND RELATED WORK

The ever-increasing size of NN models not only poses a challenge to fast and energy-efficient processing, but is a major barrier to the deployment on devices with small memory and storage, which calls for effective solutions for efficient model compression and operation count reduction. In this section, we overview some key notions related to BNNs and BNN model compression.

2.1 Binary Neural Networks

The high error resilience of NNs allows for aggressive quantization for computation at reduced precision such as fixed-point or ternary weights instead of complex full precision [12, 15]. BNNs are an extreme case of quantized NNs, where weights and activations are restricted solely to two values, +1 or -1 [11]. This binarization leads to a simplification of multiply-and-accumulate (MAC) operations, which is the most fundamental but expensive aspect of the convolutional operation, to extremely simpler XNOR and popcount operations. This leads to a significant reduction in power consumption and model size. The complexity of a BNN is measured by the number of binary operations (BOPs), instead of the number of floating-point operations (FLOPs).

A key observation is that the derivative of the binarization function at all spots is zero or undefined, making backpropagation gradient calculation impossible. Therefore, the straight through estimator is used to allows the gradient to pass exactly as an identity, generating a *pseudogradient* [2, 11]. Also, having only binary values for weights, it is impossible to distinguish distinct magnitudes between the weights in BNNs. Thus, traditional magnitude-based weight pruning is ineffective, as there is no way to determine which weight affects classification accuracy more.

2.2 Flip Frequency-based Channel Shrinking

For the weight pruning of a BNN model, a new indicator of weight significance that substitutes the weight magnitude is required. A recent work has proposed to exploit the amount of weight flips (+1 to -1 or vice versa) that occur during training [17]. In this work, they conjecture that the weights can be determined as "unstable" if they flip frequently during training. Unstable weights are considered to have little contribution in the minimization of loss within the network. When the final stage of training is near (i.e., when the loss is stabilizing), the occurrence of flips is counted for each weight kernel as f. If the number of weight flips is above a predetermined threshold, the corresponding weight is determined as negligible and thus prunable. The portion of the prunable weights represents the portion of channels that can be potentially removed. Therefore, the number of channels is reduced by the portion of the prunable weights, and the entire BNN is retrained. This is repeated until the predetermined accuracy threshold has been reached.

2.3 Latent Weights in BNN

In a BNN model, the optimizer cannot directly compute the gradients required to update the weight kernels during backpropagation because the gradient of the sign function is zero almost everywhere. Therefore, a real-valued weight vector, $\tilde{w}$, is used instead of the binary weights for training [2, 11]. Also called the *latent weight* [9], it is used to calculate the pseudogradient during backpropagation. During the forward pass, the binarized weights, w_{bin}, is simply the sign of the latent weight:

$$w_{bin} = \text{sign}(\tilde{w}) = \begin{cases} +1 & \text{if } \tilde{w} \geq 0 \\ -1 & \text{if } \tilde{w} < 0 \end{cases}. \quad (1)$$

The sign and magnitude can be thought of separately as follows [9]:

$$\tilde{w} = \text{sign}(\tilde{w}) \cdot |\tilde{w}| =: w_{bin} \cdot m, w_{bin} \in \{-1, +1\}, m \in [0, \infty). \quad (2)$$

Since there now exists a magnitude value of the latent weight, m, different techniques typically reserved for floating-point models can now be applied to BNNs. Weights build inertia m over time. The higher the inertia, the stronger the gradient signal that is required in order to make the weight flip. Weights in the forward pass can only flip and not adjust their magnitude, unlike their floating-point counterparts. However, in the backpropagation stage, m for each latent weight can adjust during each training epoch, distinguishing individual weights in the kernel from one another. This real-valued vector allows for optimization methods to be applied to the BNN. Each BNN model that is trained contains the pseudogradient information along with latent weight information.

3 PROPOSED LATENT WEIGHT-BASED PRUNING

We propose a new method to prune BNN models based on latent weights that dramatically reduces the model size and operation count, while maintaining accuracy. Specifically, we address the challenges in BNN pruning mentioned in Section 2.1: i) identify which layer should be pruned and determine how heavily it should be pruned, and ii) select weight kernels within the identified layer to be pruned.

3.1 Design Flow Overview

We first describe our pruning method in which a BNN model is pruned based on latent weights. Unlike flip frequencies [17] which are an "indirect" significance metric induced from the latent weights, latent weight-based pruning offers a more "direct" indicator of significance since the magnitude of the latent weight drives the inertia of the weight flipping. This enables us to use the source of

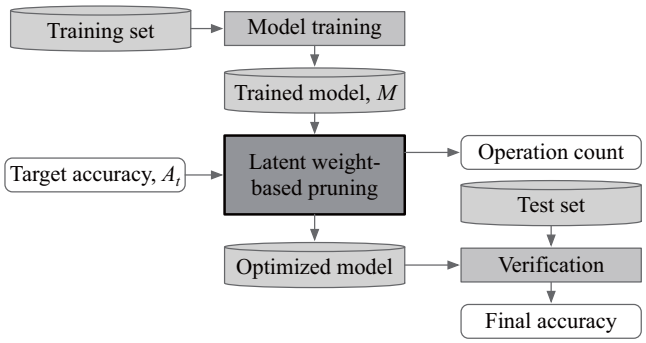

Figure 1: Proposed latent weight-based pruning method integrated in the model optimization flow.

weight kernel optimization, which offers additional granularity as we can tune pruning of real-valued weights.

The overall model optimization flow around the proposed latent weight-based pruning is presented in Figure 1. During training, we initialize all pruning percentages from zero and begin iterative pruning. From zero, we begin pruning on the least accuracy-responsive layer by increasing the pruning percentage on each layer iteratively. We prune each layer to a pre-defined accuracy threshold and prune the next least accuracy-responsive layer afterward. Our iterative pruning ends when we no longer can prune and maintain accuracy above the threshold. The following subsections describe the latent weight-based pruning highlighted in Figure 1 and its subroutines of the algorithm in detail.

3.2 Iterative Pruning Optimization

Algorithm 1 describes the main routine of the proposed latent-weight based pruning method, which is highlighted in Figure 1. The inputs to the pruning algorithm are M, A_t, and δ, where M is the trained BNN with unpruned weights, A_t is the target accuracy after pruning, and δ is the incremental increase in pruning percentage upon each iteration. Since BNNs are easily overfitted [8], A_t can be set to the accuracy of the original model before pruning, but it can also be any accuracy level that meets the application's requirement.

The pruning algorithm is performed by the iterative execution of GETSENSITIVITY to select a target layer through sensitivity analysis and PRUNELAYER to actually prune the target layer. The algorithm is iterated over each unpruned layer of M until all layers have been pruned. We first find an unpruned layer that is most robust to pruning using GETSENSITIVITY, which is described in Section 3.3, and set it as the target layer l_p. The target layer l_p is gradually pruned until further pruning violates the accuracy requirement A_t. The pruning percentage is gradually incremented by δ each time. As mentioned in Section 3.1, the initial pruning percentage for the target layer, $t(l_p)$, is initialized to zero, and it is updated after every iteration of pruning of the layer.

The value of δ should be set small enough not to miss the fine-grained optimal point of the pruning percentage, but not too big in order to minimize computational overhead. We find $\delta = 10\%$ to be reasonable for most cases. The layer-wise pruning is repeated from the least sensitive layer to the most sensitive layer. We conclude the iterative procedure once all layers have been pruned or further pruning violates the target accuracy A_t.

Algorithm 1 Latent-weight based pruning

```
1:  procedure PRUNE(M, A_t, δ)
2:      while exists an unpruned layer in M do
3:          s_max ← 0
4:          t(l_p) ← 0
5:          for each unpruned layer l do
6:              s_l ← GETSENSITIVITY(l)
7:              if s_l > s_max then
8:                  s_max ← s_l; l_p ← l
9:              end if
10:         end for
11:         while A_p > A_t do
12:             PRUNELAYER(l_p, t(l_p) + δ)
13:             t(l_p) ← t(l_p) + δ
14:             A_p ← Accuracy of pruned M
15:         end while
16:         Mark l_p as pruned
17:     end while
18:     return M
19: end procedure
```

3.3 Layer Sensitivity Analysis

When pruning convolutional layers, certain layers react with more volatility than others due to the low operation count after max-pooling or intrinsic small weight kernel size. Therefore, in order to get the most BOPs reduction without hurting accuracy, we determine the *sensitivity*, s, for every layer and prune the least sensitive layers first. For layer l, its sensitivity s_l is defined as the amount of accuracy loss, ΔA, over the operation count reduction, $\Delta BOPs$, as:

$$s_l = \frac{\Delta A}{\Delta BOPs}, \tag{3}$$

after pruning p percentage of the weights of layer l in isolation while other layers remain unpruned. The value p must be high enough to introduce accuracy instability within the network, generating a sufficient accuracy response. We find $p = 95\%$ to be reasonable in most cases to provoke a negative accuracy response within the network. This metric allows us to see which layers are less sensitive and likely to fluctuate less in accuracy when pruned. Effectively, this metric tells us how much accuracy loss we can expect a layer to contribute for a given amount of BOPs reduction. Therefore, the less sensitive a layer is, the better candidate for pruning it is.

The operation count for binarized layer l is calculated as the following:

$$BOPs_l = \left(\prod_{i=1}^{n} w_i \right) \times i_h \times i_w \tag{4}$$

where n is the dimension of the weights ($n = 4$ for convolutional layers and $n = 2$ for dense layers), w_i is the i-th index of the weight kernel, and i_h, i_w is the height and width of the output, respectively.

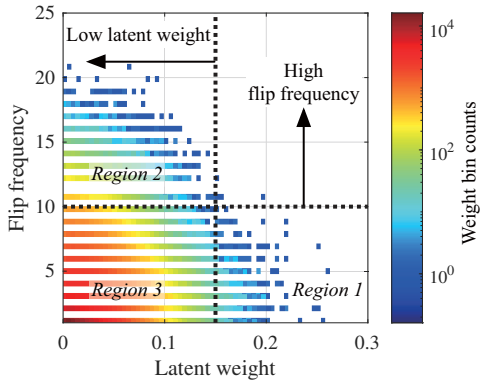

Figure 2: Comparison of the magnitude of latent weights and flips of layer 3 of BinaryNet's weight kernel.

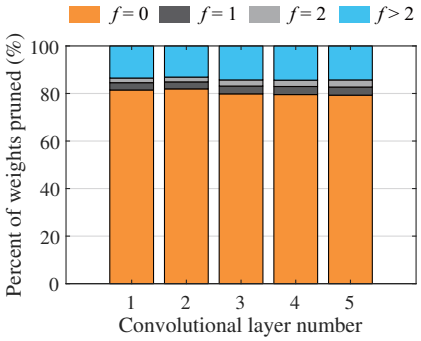

Figure 3: Discrete pruning for flip frequencies $f = 0$, $f = 1$, $f = 2$, and $f > 2$ of BinaryNet.

3.4 Pruning based on Latent Weights

Within the function PRUNELAYER(l_p, p), we prune the target layer l_p by removing the p percentile of the weight kernels with the lowest latent weight magnitude. Using latent weights offers distinct advantages over using the flip frequency. First, latent weights are a better indicator of the significance of weight kernels, which is often not correctly captured by flip frequencies. As described in Section 2.3, the larger the magnitude of the latent weights, the less likely the weight is unstable. Figure 2 shows the relationship between latent weights and flip frequencies captured from the BinaryNet as an example. It shows that the *maximum* latent weight is inversely proportional to flip frequency, but the near-zero latent weights, which are the majority of the weights (Region 3), show widely varying flip frequencies, between 0 and 21 in this example. In other words, a low flipping frequency does not always represent an important weight kernel with a high latent weight, and removing only high flip frequency weights (Region 2) may lead to ineffective pruning of weights. On the other hand, our method keeps the high latent weights (Region 1) for better pruning results, as we show in the experimental results.

Second, the real-valued nature of the latent weights allows us to perform more fine-grained pruning. We can distinguish almost every individual weight within the kernel and prune by the percentage of weights that fall below a certain threshold as opposed to pruning on discrete integer values. The ability to prune based on real-valued weights allows us to distinguish individual layers based on our sensitivity analysis as well. To illustrate the disadvantages of discretized pruning, Figure 3 shows that an overwhelming majority of flip frequencies have stabilized and are at $f = 0$, making them impossible to discern. Furthermore, flip frequencies that comprise the remainder of weights in the kernel comprise a small fraction of the overall weights. Therefore, there is no way to distinguish sensitivities and selectively choose layers to prune for a baseline flip frequency of $f > 2$. Simply pruning intermediate f values at $f = 1$ and $f = 2$ is unable to produce a sufficient accuracy response for sensitivity analysis. This is in contrast to real-valued pruning on BNNs using latent weights, where we can adjust the entire pruning threshold on a real-valued scale and effectively observe sensitivities.

Table 1: Models and datasets used in the experiments.

Model	Dataset	Bin. conv. layers	Total BOPs	Base acc.
3ConvNet	MNIST [14]	2	4.5×10^6	97.4%
BinaryNet [11]	CIFAR-10 [13]	5	5.1×10^8	80.5%
XNOR-net [20]	Imagenette [10]	5	2.9×10^9	63.0%

As a result, the proposed latent weight-based pruning achieves higher reduction in BOPs while maintaining high accuracy, as demonstrated in the following section.

4 EXPERIMENTS

In this section, we demonstrate the efficacy of our BNN pruning algorithm in comparison to the baseline unpruned networks as well as network shrinking based on flip frequency [17].

4.1 Experimental Setup

We consider three neural networks, a simple three-layer CNN with two binarized convolutional layers (3ConvNet), BinaryNet [11], and XNOR-Net [20], trained for classification of the MNIST [14], CIFAR-10 [13], and Imagenette [10] datasets, respectively, to demonstrate the proposed pruning method on different sized image inputs and network complexities. Table 1 summarizes their complexities. The models are trained using Tensorflow and each BNN is built using the Larq API [1] and pruned using the proposed method. We use batch normalization after every layer and use 20% of the training set as a validation set used to determine the accuracy. Accuracy of the final result is evaluated once on the test set previously unseen to the algorithm. We do not retrain on the validation set and only determine the accuracy threshold strictly on the validation set. Fine tuning is done post-pruning with ten epochs with a learning rate at 10% of the original training learning rate. As discussed in Section 3.2, A_t is set to the baseline accuracy of the unpruned network to enable iso-accuracy comparison.

4.2 Pruning Analysis

We describe results on all three datasets with different networks.

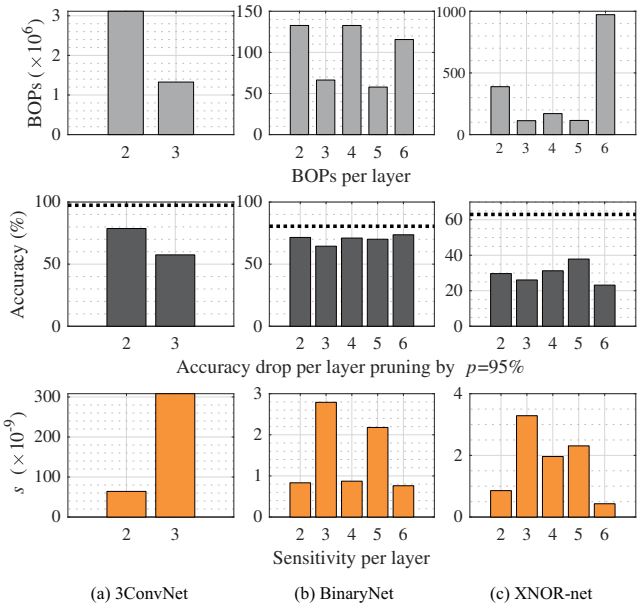

Figure 4: BOPs and sensitivities of the binarized convolutional layers of the three models. The dashed line indicates the baseline classification accuracy without any pruning.

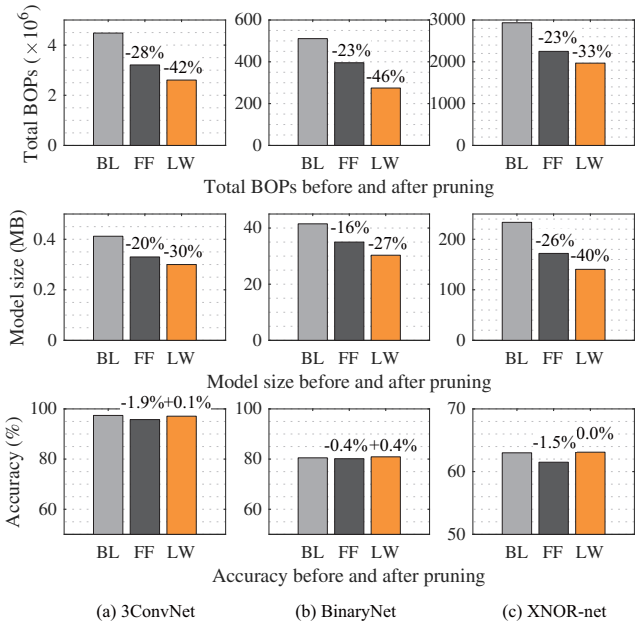

Figure 5: Comparison of total BOPs, model size, and accuracy before and after pruning. BL: baseline before pruning, FF: flip frequency-based pruning with channel shrinking [17], LW: latent weight-based pruning (proposed).

4.2.1 MNIST Classification. We first implement a small 3-layer CNN with two binary convolutional layers to classify MNIST. The first layer is a floating point activation layer with 32 output channels followed by two binary convolutional layers with 32 and 64 output channels. The small network used is immediately responsive to pruning on both layers. The initial sensitivity analysis reports that the second convolutional layer is the least sensitive to pruning with $s = 64 \times 10^{-9}$ opposed to $s = 308 \times 10^{-9}$ as displayed in Figure 4(a). We first iterate through layer 2 with $\delta = 10\%$ and then proceed onto layer 2 with the same granularity. Results indicate a 42% reduction in BOPs and 30% reduction in model size compared to the unpruned base model with no accuracy loss as demonstrated in Figure 5(a). Latent weight pruning outperforms flip frequency pruning by 19.4% in BOPs reduction and by 12.5% in model size reduction.

4.2.2 CIFAR-10 Classification. The second model classifies CIFAR-10 using the BinaryNet architecture [11]. The initial sensitivity analysis reports that convolutional layer 4 is the least sensitive to pruning with $s = 0.71 \times 10^{-9}$ in Figure 4(b). The initial sensitivity analysis indicates binary convolutional layer 6 is the least sensitive to pruning, with the lowest magnitude s value. Since each layer contributes differently to the total number of operations within the network due to max pooling layers and increasing channel output width, imbalances occur as evidenced in Figure 4(b) on the right-hand side. This proves to be beneficial in the case of layer 6, as there is a very high BOPs count within this layer, allowing great BOPs reduction at the cost of little accuracy degradation. Therefore, we prune layer 6 to its entirety until the A_t is reached after fine-tuning. At each end of each pruning iteration, we recalculate s for every unpruned layer. In BinaryNet's case with CIFAR-10 dataset, we prune layers four and six as they are the least sensitive layers. In Figure 6(a), the first three iterations with multiple pruning steps are displayed, with layers 6, 4, and 2 only being pruned once before reaching the accuracy threshold. Beyond these first two iterations, we degrade accuracy beyond the A_t threshold. Results indicate a 46% reduction in BOPs and 27% reduction in model size compared to the unpruned base model with no accuracy loss as demonstrated in Figure 5(b). Latent weight pruning outperforms flip frequency pruning by 29.9% in BOPs reduction and by 13.1% in model size reduction.

4.2.3 Imagenette Classification. We implement the XNOR-net architecture to classify the Imagenette dataset for our third network analysis [10, 20]. We initialize pruning on the fourth convolutional layer due to its sensitivity being the lowest at $s = 0.43 \times 10^{-9}$ as displayed in Figure 4(c). Layer 6 of XNOR-net contains a 6×6 convolutional filter, allowing for this layer to be significantly reduced by pruning over 80% of the original weights. Additionally, pruning layer 6 has massive implications on overall model size, since layer 6 comprises 62% of the total model storage size, allowing us to greatly reduce the model size more than the other designs. Following the pruning of layer 6, layer 5 is pruned according to it having the lowest sensitivity, which is again re-evaluated at the end of each layer pruning iteration. Within Figure 4(c), the sensitivities only indicate what layer will be pruned on the first iteration, which is layer 6. The sensitivity is recalculated at the end of each iteration, meaning that while layer 2 has a lower sensitivity than layer 5 on the first iteration, layer 2 will not necessarily be pruned before layer 5. We increase pruning at $\delta = 10\%$ and prune layer-by-layer until the accuracy threshold is reached at 63%. In Figure 6(b), the first three

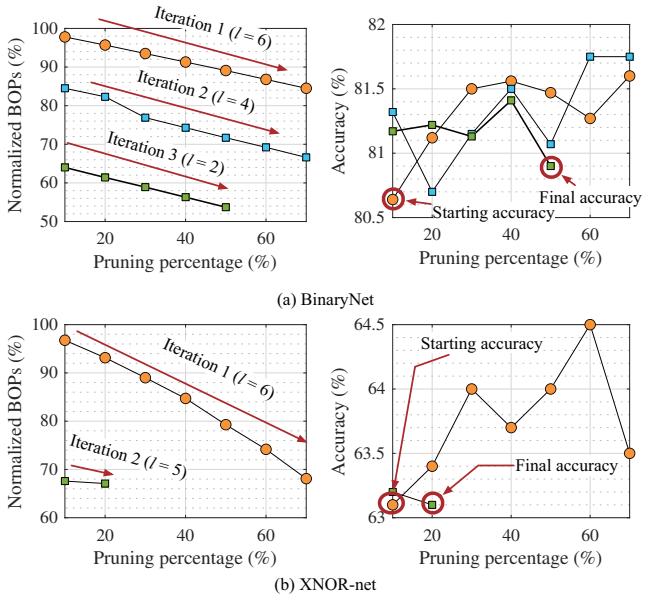

Figure 6: BOPs reduction and classification accuracy during pruning of (a) BinaryNet and (b) XNOR-net.

iterations with multiple pruning steps are displayed, with layer 1 only being pruned once before reaching the accuracy threshold.

Results indicate a 33% reduction in BOPs and 40% reduction in model size compared to the unpruned base model with no accuracy loss as demonstrated in Figure 5(c). Latent weight pruning outperforms flip frequency pruning by 13.0% in BOPs reduction and by 18.9% in model size reduction.

4.3 Discussion

We compared our method to purely using a channel shrinking method based on weight flipping frequencies described in [17]. Our method provides a distinct advantage where channel shrinking is removing complexity from the network without exploiting the sparse resilience and lottery-ticket behavior of the network. In this method, we demonstrate the importance of exploiting sparsity within the network, as only a select few connections within the network are shown to be major contributors to the final classification accuracy [5].

Our results using iterative pruning incurs additional offline costs but produce a more efficient final pruned model. These offline costs can be handled by GPUs, producing a portable model for loading onto storage and computation-constrained systems. In particular, we have greatly reduced the amount of BOPs within the network. Our goal is to take computationally demanding training and not have it be handled by the edge device. The only edge device responsibility is inference with the ported model. While the accuracy gain is mild, this unstructured pruning regularizes the network efficiently. Channel shrinking, on the other hand, does not take advantage of the inertia of the binary weights and provides no guarantee for regularization. Thus, we conclude that with our method, we produce a more efficient and accuracy-robust pruned model than the previous work.

5 CONCLUSIONS

Latent weight-based BNN pruning is a promising approach which mixes two popular neural network compression techniques: quantization and weight pruning. The classical unstructured pruning that has been used in floating point models is difficult to integrate with BNNs due to their binary nature and the lack of weight magnitude in the forward pass. We demonstrated that latent weights that exist during backpropagation are a promising alternative that allows pseudogradient weights to represent how negligible a weight is. We presented a pruning solution that is precision-tuned to each layer, querying the sensitivities of individual components of the network to prune in a coordinated manner. In particular, our method focuses on reducing computational complexity and memory storage overhead of the pruned model. Compared to the previous work of pruning binary neural networks, we achieve a lower OPs count and smaller model size. On all three datasets with three different architecture sizes, we demonstrated 33%–46% reduction in operation count and a 27%–40% reduction in model size with no accuracy loss or up to a +0.4% gain.

ACKNOWLEDGMENTS

This work was supported by NSF under awards CNS-1845469, CNS-2112562, and DGE-1747503.

REFERENCES

[1] Tom Bannink et al. 2021. Larq compute engine: Design, benchmark and deploy state-of-the-art binarized neural networks. *MLSys*.
[2] Yoshua Bengio et al. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.
[3] Tianen Chen et al. 2022. SynthNet: A High-throughput yet Energy-efficient Combinational Logic Neural Network. *ASP-DAC*.
[4] Xizi Chen et al. 2020. Tight compression: compressing CNN model tightly through unstructured pruning and simulated annealing based permutation. *DAC*.
[5] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.
[6] Jiabao Gao et al. 2021. An Approach of Binary Neural Network Energy-Efficient Implementation. *Electronics*.
[7] Song Han et al. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.
[8] Song Han et al. 2015. Learning both weights and connections for efficient neural network. *NIPS*.
[9] Koen Helwegen et al. 2019. Latent weights do not exist: Rethinking binarized neural network optimization. *NIPS*.
[10] Jeremy Howard. 2019. Imagewang. https://github.com/fastai/imagenette/
[11] Itay Hubara et al. 2016. Binarized neural networks. *NIPs*.
[12] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*.
[13] Alex Krizhevsky et al. 2014. The CIFAR-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*.
[14] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/
[15] Fengfu Li et al. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711*.
[16] Yixing Li et al. 2020. Build a compact binary neural network through bit-level sensitivity and data pruning. *Neurocomputing*.
[17] Yixing Li and Fengbo Ren. 2020. BNN Pruning: Pruning Binary Neural Network Guided by Weight Flipping Frequency. *ISQED*.
[18] Ji Lin et al. 2020. MCUNet: Tiny deep learning on IoT devices. *NIPS*.
[19] Mahdi Nazemi et al. 2019. Energy-efficient, Low-latency Realization of Neural Networks Through Boolean Logic Minimization. *ASP-DAC*.
[20] Mohammad Rastegari et al. 2016. XNOR-Net: Imagenet classification using binary convolutional neural networks. *ECCV*.