Partial Binarization of Neural Networks for Budget-Aware Efficient Learning

Udbhav Bamba[†,*], Neeraj Anand[✢,*], Saksham Aggarwal[†,*], Dilip K. Prasad[‡] and Deepak K. Gupta[†*]
[✢] Nyun AI, India
[†]Transmute AI Lab, India
[‡]UiT The Arctic University of Norway, Norway
{ubamba98,neerajanandfirst,sakshamaggarwal20}@gmail.com

Abstract

Binarization is a powerful compression technique for neural networks, significantly reducing FLOPs, but often results in a significant drop in model performance. To address this issue, partial binarization techniques have been developed, but a systematic approach to mixing binary and full-precision parameters in a single network is still lacking. In this paper, we propose a controlled approach to partial binarization, creating a budgeted binary neural network (B2NN) *with our* `MixBin` *strategy. This method optimizes the mixing of binary and full-precision components, allowing for explicit selection of the fraction of the network to remain binary. Our experiments show that B2NNs created using* `MixBin` *outperform those from random or iterative searches and state-of-the-art layer selection methods by up to 3% on the ImageNet-1K dataset. We also show that B2NNs outperform the structured pruning baseline by approximately 23% at the extreme FLOP budget of 15%, and perform well in object tracking, with up to a 12.4% relative improvement over other baselines. Additionally, we demonstrate that B2NNs developed by* `MixBin` *can be transferred across datasets, with some cases showing improved performance over directly applying* `MixBin` *on the downstream data.* [1]

1. Introduction

Convolutional neural networks (CNNs) have led to several breakthroughs in the field of computer vision and image processing, especially because of their capability to extract extremely complex features from the images. However, these deep CNN models are extremely computation-hungry and require significant power to process. For example, a ResNet-18 classification model comprises 1.9 million parameters, each represented in full-precision using 32-bits, and accounts for a total of 1.8 billion floating point operations (FLOPs) for the ImageNet dataset [1]. For most of the problems, these deep CNN models are overparameterized, and there is enormous scope of reducing their sizes with minimal to almost no drop in the performance of the models. The popular approaches for effective model compression include removing the non-important set of parameters or channels (pruning) [2], distilling knowledge of the dense teacher network into a light-weight student network (distillation) [3], converting 32-bit representations of the parameters to lower bit representations (quantization) [4], and transforming the network weights to 1-bit representations (binarization) [5].

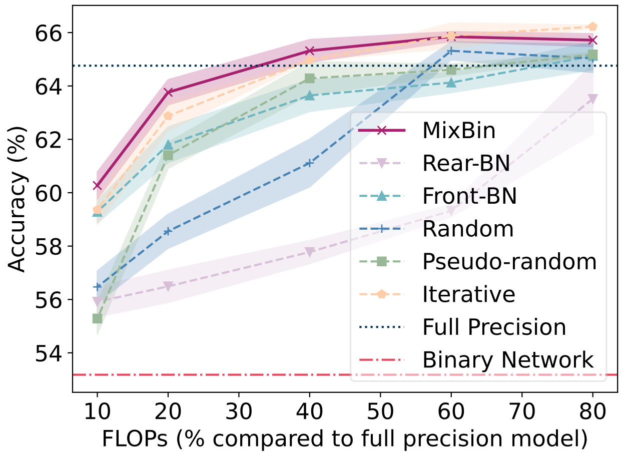


Figure 1. **Performance comparision of B2NNs obtained using `MixBin` vs. the various trivial approaches**. The B2NNs are constructed using cResNet-20 architecture on CIFAR-100 datasets. Here, *Rear-BN* and *Front-BN* represent B2NNs constructed with binary layer placed in the rear and front parts of the network, respectively. *Pseudo-random* involves modifying 30% of the MixBin generated B2NNs. The Full Precision network and Binary Network have 4.14×10^7 and 1.93×10^6 FLOPs respectively.

Among the methods outlined above, binarization is very

[*]Equal Contribution
[1]Code is publicly available at https://github.com/transmuteAI/trailmet

effective in drastically reducing the size of the model and increasing the inference speed. In its basic form, binarization involves changing all the weights and activations to 1-bit representation and implementing the convolutions with bitwise XNOR operations [6]. However, due to the significantly reduced representation, the performance of the binarized network is significantly lower than its full-precision counterpart. To circumvent this, several approaches exist such as binarizing only the weights and keeping the activations as full-precision [7], parallely stacking multiple binarized layers [8], using special layers such as squeeze-and-excitation blocks [9] and retaining skip connections as full-precision modules in a ResNet-type binary network [10]. All the above methods exploit partial binarization of the network reduce the performance gap between full precision network and binarization. However, there still does not exist a systematic approach to perform intermediate levels of compression in a more controlled sense. A solution as such would provide the flexibility of analyzing the drop in the performance of the model at different levels of model compression, thereby allowing to choose a right balance between the size of the binararized model and its performance. Beyond this, such an approach would provide the control on using binarization to perform hardware-specific compression, thereby allowing the compressed model to exploit the full computational power budget of the target hardware.

An alternative to using partial binarization could be quantization, where based on the desired extent of compression, the choice of precision for the target network can be made. However, due to its ability to replace the matrix multiplications with XNOR operations, binarization can achieve a higher extent of FLOP reduction compared to an equivalent amount of quantization in terms of similar number of effective parameters. Moreover, binarization itself can be looked at as a modified form of aggressive quantization, and it is worth understanding how it would fair.

In this paper, we propose a paradigm to perform partial binarization of neural networks in a controlled sense, thereby constructing *budgeted binary neural network (B2NN)*. Our B2NN approach relies on identifying the right set of convolutional layers of a network that should be binarized, and the rest of the network is retained as full-precision. A straightforward approach to select layers to binarize is through random sampling, and we experimentally demonstrate that this is not very effective. While the network itself can be made very light, the performance of the compressed variant deteriorates significantly. Through various trivial baselines, we numerically demonstrate that binarizing different parts of a CNN has very different effect on the performance of the compressed model, thus making it important that the right set of target layers are identified for building the right B2NN. This is shown in Figure 1, and we see that for a similar computational budget, keeping the later layers of a network as full-precision boosts its performance significantly, while more binarization towards of the end of the network architecture has an adverse effect. We discuss this aspect further in section 4 of this paper.

To overcome the challenge outlined above, we present MixBin, a smart selection strategy that constructs B2NN through optimized mixing of the binary and full-precision components. MixBin allows to explicitly choose the fraction of the network to be kept as binary, thereby presenting the flexibility to approximately adapt the inference cost to a prescribed budget. The core of our selection strategy lies in effectively analyzing how sensitive the performance of the network is with respect to the different layers. Based on our initial observations, we have formulated two variants, $\texttt{MixBin}_{Loss}$ and $\texttt{MixBin}_{Grad}$, and the related details are described in section 3.3 of this paper. We conduct several experiments and demonstrate in several different scenarios that the resultant compressed networks achieved using MixBin are superior over all the baselines chosen in our study, which includes some of the popularly used model compression methods as well.

Contributions. The contributions of this paper can be summarized as follows.

- We introduce the concept of budgeted binary neural networks (B2NNs), compressed variants of the dense full-precision models obtained through partial binarization. B2NN relies on effectively identifying the right set of layers that are to be retained as binary or full-precision.
- We present MixBin, a search strategy to compress a given full-precision network through partial binarization in an optimized sense. The inherent design of MixBin facilitates budgeted binarization, allowing to develop light-weight models that maximally exploit the available compute resources. We demonstrate through numerical experiments that B2NNs obtained from our MixBin strategy are significantly better than those obtained from random selection or even iterative greedy search over the network layers.
- The efficacy of MixBin is demonstrated on multiple datasets and tasks for various budget scenarios, and the resultant models obtained from MixBin are consistently superior over those obtained with popular model compression methods chosen as baselines in this study. Performance results on ImageNet demonstrate the effectiveness of MixBin for large datasets, and we also show that it works well for the downstream task of object tracking.

2. Related Work

Neural networks are often known to be overparameterized [11]. This leads to undesired additional latency when deploying these models in production with little effect on the model accuracy. There exist several works that focus on addressing this issue. First among these is through inducing efficient components in the neural network design, such as using bottleneck blocks [12], replacing the 3×3 convolutions with 1×1 [13], using depthwise-separable convolutions [14] and employing neural architecture search [15–17]. Another approach to design efficient networks involves distilling the information of large networks into smaller networks (knowledge distillation) [3, 18–21]. Further, there exist works that aim at identifying the undesired or less desired weights or filters of a network and removing them. Selection criteria for pruning include ranking scores based on L1/L2 norms of parameters [11,22,23], gradients derivatives [24, 25], learnable parameters [2, 26, 27] and pruning of network once at intialization prior to training [28, 29] . An equally effective direction of efficient network design is through quantization of the weights and/or activations of a network to represent it with a reduced number of bits [4, 30–32].

Quantizing a given network refers to representing each weight or activation with reduced number of bits, such as half-precision (16-bits) or 8-bits. An extreme case of quantization is network binarization where the 32-bit representations are directly scaled down to 1-bit each [5]. Due to the replacement of the conventional convolution operation with a bitwise XNOR operation, a significant computational gain is observed. Further, adding the channelwise scaling on the binary weights allows to scale BNNs to largescale datasets such as ImageNet [6] . Although binarization is an effective technique for network compression, it leads to a significant reduction in network representation which results in a significant decrease in performance. There exist several works that attempt to find a right balance between model performance and the extent of compression in the model. For example, ABCNet proposes to stack multiple parallel layers together to use multiple binary layers to increase the representation capability of the network [8]. In BiRealNet [10], skip connections are represented as real-valued and it has been shown to boost the performance. Other recent binarization methods that improve performance include Reactnet [33], IR-Net [34] and SA-BNN [35].

Most of the approaches listed above attempt to find representations in between a fully binary network and real-valued one. However, there is no straightforward method to design networks comprising binary components that make efficient use of the available computational memory and delivering maximum possible performance. The closest towards this goal is the hybrid binary network [36] that performs selective binarization through locally converting the activations to full-precision and retaining the rest of the network as binary . However, this approach provides limited flexibility in terms of full exploitation of the mixing between binary and full-precision components. There also exists layer selection methods for quantization designed to combine different precisions layers together in a network [37–39]. These use either the difference of network weights, their gradients or the Hessians to decide which layers to choose for reduced precision and the extent of reduction. We discuss these methods further in Section 4 and present a comparison of these methods with `MixBin`.

3. Proposed Approach

3.1. Background

Binary neural networks (BNNs), also referred to as 1-bit neural networks, use binary weight parameters and binary activations for the intermediate layers of the network, excluding the first layer. Sign$(\cdot)$ function is used to convert real-valued weights/activations to their binary counterparts and the conversions can be mathematically stated as

$$\begin{aligned} a_b = \text{Sign}(a_r) &= \begin{cases} -1 \text{ if } a_r < 0 \\ +1 \text{ otherwise} \end{cases} \\ w_b = \text{Sign}(w_r) &= \begin{cases} -1 \text{ if } w_r < 0 \\ +1 \text{ otherwise} \end{cases} \end{aligned} \tag{1}$$

where a_r and w_r denote the real-valued (full-precision) activations and weights, and a_b and w_b the corresponding binary variants.

Compared to the full-precision model where 32-bit representations are used for every parameter, BNNs, with their 1-bit representations, can lead up to 32× memory saving. Further, since the activations are also chosen as binary, the convolution $(*)$ operation is implemented as a bitwise XNOR $(\oplus)$ operation and a bit-count operation. It is represented as

$$\mathbf{a}_r * \mathbf{w}_r \approx \boldsymbol{\alpha} \odot (\mathbf{a}_b \oplus \mathbf{w}_b) \tag{2}$$

where $\boldsymbol{\alpha} \in \mathbb{R}_{+}^{c_{out}}$ contains the channelwise scaling factors and $\odot$ denotes the elementwise multiplication operation. For $\mathbf{w}_r \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times k_{\text{h}} \times k_{\text{w}}}$, the scaling factor $\alpha_i \in \boldsymbol{\alpha}$ can be mathematically represented as

$$\alpha_i = \frac{1}{n} \sum \mathbf{w}_r^{(i,:,:,:)} \tag{3}$$

denoting summation of the matrix along all dimensions except c_{out}, and $n = c_{in} \times k_h \times k_w$. Here, c_{in}, k_h and k_w denote the input channel dimension, kernel height and width, respectively. For more details related to the scaling, see [6].

3.2. Budgeted binary neural network (B2NN)

Although BNN leads to significant memory and computation gain, it has been experimentally demonstrated that the performance of BNNs can be significantly lower than their full-precision counterparts. Clearly, reducing 32-bits to 1-bit in all parts of the network is not the effective way, and budgeted binary neural network alleviates this issue. Among the various layers of a given CNN for example, B2NN identifies the right set of layers that are to be converted to 1-bit representation with minimal compromise in model performance, and the rest are retained as full-precision. Below we provide the mathematical description of B2NN.

Let $\mathcal{F} : \mathbb{R}^d \rightarrow \mathbb{R}^c$ denote a neural network comprising a set of weights $\mathbf{W}$, activations $\mathbf{A}$, and input and output layers. For $\mathcal{F}$ comprising n hidden layers, this implies, we have $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n, \mathbf{w}_{n+1}\}$ and $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n\}$. Note here that $\mathbf{w}_{n+1}$ performs a fully-connected mapping between the output of the final convolutional layer and the output of the model. During binarization, it is common for even fully-binarized networks to retain $\mathbf{w}_1$, $\mathbf{w}_{n+1}$ and $\mathbf{a}_n$ as full-precision [10, 33], and we follow a similar convention . Thus, for methods that perform complete binarization, $\mathbf{w}_i \; \forall \; i \in [2, n]$ and $\mathbf{a}_i \; \forall \; i \in [1, n-1]$ are converted from full-precision to binary. However, as stated earlier, this dips the performance of the resultant BNN significantly, and alternatively some of the works retain $\mathbf{a}_i$ as full-precision.

B2NN couples $\mathbf{a}$ and $\mathbf{w}$ together as $\boldsymbol{\theta}_i = \{\mathbf{a}_i, \mathbf{w}_{i+1}\}$, referring them as one layer, and performs binarization on a subset $\mathbf{\Phi} \subset \mathbf{\Theta}$, where $\mathbf{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{n-1}\}$. The generic mathematical problem that we solve with B2NN can be stated as follows.

$$\begin{aligned} \mathbf{\Phi}^* = \underset{\mathbf{\Phi} \subset \mathbf{\Theta}, \mathbf{W}}{\text{argmin}} \; & \mathcal{L}(\mathcal{F}(\mathbf{\Phi}, \mathbf{\Theta} - \mathbf{\Phi}, \mathbf{x}), \mathbf{y}) \\ \text{s.t.} \quad & \mathcal{B}(\mathbf{\Phi}, \mathbf{\Theta} - \mathbf{\Phi}) \leq \mathcal{B}_0, \end{aligned} \tag{4}$$

where $\mathbf{\Phi}^*$ denotes the optimized subset of layers that are binarized, and $\mathcal{L}(\cdot)$ denotes the function to be minimized on the dataset $(\mathbf{x}, \mathbf{y})$ when making this selection. Further, $\mathcal{B}(\cdot)$ denotes the budget function and $\mathcal{B}_0$ is the prescribed limit. In this paper, we use FLOPs budget for compressing the networks.

Effect of binarizing different parts of a network. The performance of the constructed B2NN model depends heavily on the choice of $\mathbf{\Phi}$. To demonstrate the importance of selection, we consider several trivial approaches in this study to construct B2NNs, where the goal is to select k out of $n-1$ layers to be binarized, such that the performance of the resultant B2NN is maximized to the largest possible extent while also satisying the budget constraint as stated in Eq. 4. We provide a brief description of the various trivial approaches below followed by detailed formation of our `MixBin` approach in Section 3.3.

Random selection. As the name suggests, this approach does not involve any smart strategy in the selection and the process of selecting k out of $n-1$ layers to be binarized is completely random.

Front- or Rear-BN selection. In this selection strategy, $\mathbf{\Phi}^*$ is sampled sequentially from the front or rear parts of the full-precision network, respectively. For Front-BN selection, this implies selecting $\mathbf{\Phi}^*$ as $\mathbf{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k\}$. Similarly, for Rear-BN selection , we have $\mathbf{\Theta} = \{\boldsymbol{\theta}_{n-k}, \boldsymbol{\theta}_{n-k+1}, \ldots, \boldsymbol{\theta}_{n-2}, \boldsymbol{\theta}_{n-1}\}$.

Iterative selection. It is a greedy selection process based on iterative search strategy designed to identify the right layers of any given network to be converted to binary or full precision, one-by-one. For the k out of $n-1$ layers to be binarized, the j^{th} step of binarization, where $j \in [1, k]$, can be stated as finding the optimal layer $\boldsymbol{\theta}^* \in \mathbf{\Theta}_{(j)}$ to be binarized. It can be mathematically stated as:

$$\begin{aligned} \boldsymbol{\theta}^*_{(j)} = \underset{\boldsymbol{\theta} \subset \mathbf{\Theta}_{(j)}, \mathbf{W}}{\text{argmin}} \; & \mathcal{L}(\mathcal{F}(\boldsymbol{\theta} + \mathbf{\Phi}_{(j)}, \mathbf{\Theta}_{(j)} - \boldsymbol{\theta}, \mathbf{x}), \mathbf{y}) \\ \text{s.t.} \quad & \mathcal{B}(\boldsymbol{\theta} + \mathbf{\Phi}_{(j)}, \mathbf{\Theta}_{(j)} - \boldsymbol{\theta}) \leq \mathcal{B}_0, \end{aligned} \tag{5}$$

where $\mathbf{\Theta}_{(j)} = \mathbf{\Theta} - \mathbf{\Phi}_{(j)}$. Here, $\mathbf{\Phi}_{(j)}$ denotes the layers that have already been binarized in the previous $j-1$ steps and is defined as $\mathbf{\Phi}_{(j)} = \{\boldsymbol{\theta}^*_{(1)}, \boldsymbol{\theta}^*_{(2)}, \ldots, \boldsymbol{\theta}^*_{(j-1)}\}$, where $\boldsymbol{\theta}^*_j$ denotes the optimal layer chosen at the j^{th} step of binarization to obtain B2NN. For calculating $\boldsymbol{\theta}^*_j$, we perform brute search over all elements of $\mathbf{\Theta}_{(j)}$ and choose the layer, which when binarized, maximizes the performance of the intermediate B2NN model. Additional details and the pseudo-code related this approach can be found in the supplementary material.

Pseudo-random selection. This selection strategy involves identifying the $\mathbf{\Phi}^*_{(j)}$ layers to be binarized and then rather than binarizing these layers, a subset of them is replaced by another subset of same size randomly sampled from $\mathbf{\Theta} - \mathbf{\Phi}$. It can be expected that if the process of selecting $\mathbf{\Phi}^*_{(j)}$ is properly optimized with respect to performance of the resultant B2NN, the performance of the network obtained from pseudo-random selection would be relatively sub-optimal.

As shown in Figure 1, mixing full-precision and binary layers using different strategies yields very different results, and a wrong selection approach can lead to completely sub-optimal model performance. For example, the performance of Rear-BN is far inferior to even the random selection strategy, indicating that the later parts of the network should not be preferred for binarization. Among the various methods described above, we observe that Iterative Selection works the best, although being lower than our `MixBin` approach. However, the iterative selection approach is computationally very expensive which limits its adoption for large networks. For an extremely low budget of 10%, we observe that almost all the methods result in poorly performing networks, and Pseudo-random ranks the lowest. Clearly, the trivial layer selection strategies described above are not well suited for constructing B2NNs, especially when it comes to extreme model compression. This shows the importance and need of a systematic way to construct B2NNs such that the maximal performance of the original network can be retained.

```
Algorithm 1: MixBin Approach
  Given  : Budget value $\mathcal{B}_0$;
           Pre-trained network weight $\mathbf{\Theta}$;
           Neural Network $\mathcal{F}$; Training data $\mathcal{D}$;
  Output: Binarized weight $\mathbf{\Phi}$
  $K \leftarrow \{\}$;
  for $\boldsymbol{\theta} \in \mathbf{\Theta}$ do
      $(\mathbf{x}, \mathbf{y}) \leftarrow \mathcal{D}$;
      // Binarize both activation and
         weights of $\boldsymbol{\theta}$
      $\hat{\boldsymbol{\theta}} \leftarrow$ Binarize($\boldsymbol{\theta}$);
      // Pass $\mathbf{x}$ through $\mathcal{F}$, B2NN with a
         single binary layer $\tilde{\boldsymbol{\theta}}$
      $\tilde{\mathbf{y}} \leftarrow \mathcal{F}(\hat{\boldsymbol{\theta}}, \mathbf{\Theta} - \boldsymbol{\theta}, \mathbf{x})$
      // Calculate Binarization
         coefficient for $\boldsymbol{\theta}$
      $\kappa \leftarrow$ MixBin($\mathbf{y}, \tilde{\mathbf{y}}, \hat{\boldsymbol{\theta}}, \mathbf{\Theta} - \boldsymbol{\theta}$)
      // Add tuple $(\kappa, \boldsymbol{\theta})$ to set $K$
      $K \leftarrow$ push($\{\kappa, \boldsymbol{\theta}\}$)
  end
  // Sort set $K$ based on $\kappa$
  $K \leftarrow$ Sort($K|\kappa$)
  // Select $\mathbf{\Phi}$ from $K$ for the
     prescribed budget $\mathcal{B}_0$
  $\mathbf{\Phi} \leftarrow$ Select($K|\mathcal{B}_0$)
```

3.3. MixBin

MixBin refers to the strategy of mixing binary and full-precision components in a network in an efficient and effective manner. The approach of MixBin is described in Algorithm 1. Unlike the greedy selection approach discussed earlier, MixBin identifies the set of layers to be binarized, $\mathbf{\Phi}^*$, in one single pass eliminating the requirement for any model updates via backpropagation. It involves first calculating the binarization coefficient κ with respect to every layer. To compute $\kappa_i \ \forall \ \kappa_i \in [1, n-1]$, only the i^{th} layer is converted to binary and the performance of the resultant network is computed. Note that κ can be any generic function such that magnitude of κ_i correlates with the extent to which the i^{th} layer would be a preferable choice for binarization. In this paper, we choose it to be a function of drop in performance of the network due to binarization of the i^{th} layer and/or the change in the gradient of the loss with respect to this layer.

For the selection of $\mathbf{\Phi}^*$ as described in Eq. 4, the values of κ associated with all the layers are analyzed and a selection is made. In this paper, we present two variations of MixBin that differ how κ is calculated. These are described below.

MixBin$_{Loss}$ The premise of this formulation assumes that if a specific layer is favored over the others for binarization, converting this layer from full-precision to binary, while maintaining all other layers at full-precision, would result in minimal performance degradation. In other words, the drop in performance of the model due to binarization of a certain layer can be considered independent of the others and can be used as a direct measure of how preferred a certain layer is for binarization. Based on this, the binarization coefficient is defined as

$$\begin{aligned}\kappa_j^{Loss} = \mathcal{L}(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}) - \\ \mathcal{L}(\mathcal{F}(\hat{\boldsymbol{\theta}}_j, \mathbf{\Theta} - \hat{\boldsymbol{\theta}}_j, \mathbf{x}), \mathbf{y})\end{aligned} \tag{6}$$

The resultant values of κ are then sorted in ascending order and the layers corresponding to the first k sorted values of κ are chosen ensuring that the extent of binarization complies with the predefined budget defined for the B2NN.

MixBin$_{Grad}$ This formulation is based on the hypothesis that beyond the drop in performance of the model, the inertness of the resultant network also plays an important role in deciding the importance of the layers for binarization. Based on this, we also include the L_1 norm of the gradient of the resultant loss with respect to the parameters of the j_{th} layer. The resultant binarization coefficient κ^{Grad} can be stated as

$$\boldsymbol{\kappa}_j^{Grad} = \boldsymbol{\kappa}_j^{Loss} \times \left\| \mathcal{L}(\mathcal{F}(\hat{\boldsymbol{\theta}}_j, \mathbf{\Theta} - \hat{\boldsymbol{\theta}}_j, \mathbf{x}), \mathbf{y})(\widehat{\boldsymbol{\theta}_j}) \right\| \tag{7}$$

Intuition for the addition of the gradient term is that once a layer has been binarized, it should have have lower tendency to influence the performance of the model, thereby exhibit increased inertness.

4. Experiments

4.1. Experimental Setup

We demonstrate the efficacy of MixBin on the tasks of image classification and object tracking. For image classification, we conduct experiments to compress CIFAR-ResNet20 (referred further as cResNet-20) on CIFAR-100, MobileNet and ResNet-18 on TinyImageNet and ResNet-18 on ImageNet-1K dataset. For Object tracking, we use ResNet-18 model, and train and compress the tracker on GOT-10K dataset. The choice of smaller architectures like cResNet-20 and MobileNet help us to ensure that the architectures do not overfit on simpler datasets like CIFAR-100 and TinyImageNet. Further the choice of ImageNet-1K and GOT-10K help to demonstrate that MixBin is scalable to complex datasets.

FLOPs calculation. We calculate FLoating point OPerations (FLOPs) based on the code publicly available at https://github.com/Swall0w/torchstat/. For the calculations, batch size of 1 is assumed. For B2NNs, total FLOPs of the network is equal to the sum of FLOPs of the full-precision layers and BOPs (Binary OPerations) for binary counterpart. BOPs are determined using the methodology introduced in BiRealNet [10]. This involves dividing FLOPs by 64 since modern CPUs can execute bitwise XNOR operations and bit-counting concurrently in groups of 64.

4.2. Baselines

To demonstrate the efficacy of MixBin, we set up multiple competitive baselines. These include the random and greedy selection strategies as described in Section 3.2, as well as adaptations of works which propose methods for optimal selection of layers based on different bit-widths. We adapt these methods for our use case as baselines in which we either keep the whole layer binary or full precision. Note that while we explain the methods below in terms of quantization, as in their original formulation, we use the resultant indicators to identify the right layers for binarization.

BNAS [37]. This approach incorporates amplitude loss function as part of the optimization process. This loss function is formulated as the L_2 difference between the full precision weights and the binary weights. We employ the amplitude loss as a criterion for selecting layers in the network, where a lower value indicates a higher inclination towards binarization.

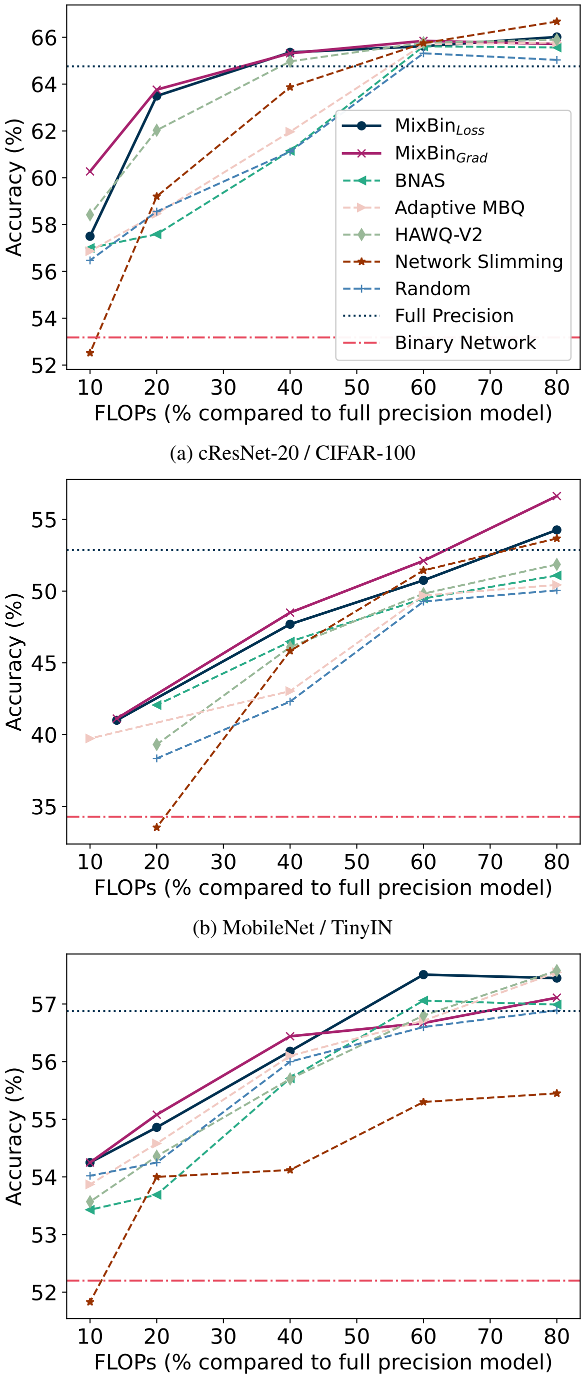
Adaptive MBQ [38]. This approach uses Taylor expansion to build a metric for quantifying the loss sensitivity introduced by quantization. The metric guides on the extent to which each layer should be quantized, and a lower value of their proposed metric indicates a higher level of quantization.

HAWQ-V2 [39] uses the sensitivity of Hessian trace of weights, scaled by a perturbation of quantization. By considering second-order information stored in the Hessian, it identifies layers that are more sensitive to quantization. A lower value of the metric indicates a higher level of quantization.

Network Slimming [2]. Additionally, we also compare our method with a popular network pruning approach, called network slimming [40]. This approach involves adding an L_1 penalty term on the scaling parameters of each Batch Normalization layer which induces sparsity in the network.

4.3. Results

Performance on simple classification datasets. Figure 2 presents the performance curves obtained with MixBin as well as the chosen model compression baselines on CIFAR-100 and TinyImageNet datasets. Baselines here include those mentioned in Section 4.2, as well as random selection. Across all combinations of the datasets and mod-



(a) cResNet-20 / CIFAR-100

(b) MobileNet / TinyIN

(c) ResNet-18 / TinyIN

Figure 2. Performance comparison of MixBin with B2NNs obtained using random and competitive layer selection algorithms, as well as pruning, for three different model and data combinations.

Table 1. Performance scores for the ResNet-18 architecture on ImageNet-1K datasets for three compression methods: Network Slimming, HAWQ-V2, and MixBin (ours). The Full precision network has a total of 1.82×10^9 FLOPs, while the binary network generated using the Bi-RealNet method has 1.71×10^8 FLOPs. The term "budget" refers to the percentage of FLOPs that remain after applying the respective compression method.

Method	Budget(%)	Acc.(%)↑	FLOPs(%)↓
Full Precision Network	-	65.66	100
Binary Network	-	55.54	9.39
Network Slimming	50	63.58	50
HAWQ-V2		62.25	53.16
MixBin$_{Grad}$		64.12	53.16
MixBin$_{Loss}$		**65.38**	**53.16**
Network Slimming	15	34.58	15
HAWQ-V2		56.20	15.64
MixBin$_{Grad}$		57.26	15.64
MixBin$_{Loss}$		**57.44**	**15.64**

els, MixBin consistently outperforms all the baselines. At higher budgets, performance achieved with pruning are comparable to those achieved by MixBin. However, when low budgets are used, the performance of the former deteriorates to a level even lower than that of the binary model. Interestingly, we observed that MixBin at budgets of 60% or higher yields results that are comparable to, or even superior to, the original full precision model. This clearly shows that MixBin leads to compressed models that generalize better, leading to reduced overfitting and thereby improved performance on the evaluation set.

Performance on ImageNet-1K. To further demonstrate the efficacy of MixBin, we study its compression capability on large-scale classification dataset of ImageNet-1K using ResNet-18 architecture. We compare our results to those obtained by HAWQ-V2 and Network Slimming at budget levels of 15% and 50%. The performance of each method, along with the corresponding achieved FLOPs ratio compared to the full precision network, are presented in Table 1. Pruning exhibited a significant decline in scores at lower budgets, whereas binarization did not experience such a drop. Among the evaluated methods, MixBin$_{Loss}$ demonstrated the highest performance, followed by MixBin$_{Grad}$ and HAWQ-V2. Interestingly, MixBin$_{Loss}$ was able to maintain performance comparable to the full precision network at 50% budget, while the performance of the other methods dropped by 2-3%.

Note that FLOPs of B2NNs are not always met with respect to the corresponding budget since they operate at layer level, unlike structured pruning methods. However, this characteristic can be advantageous for B2NNs as they have a more structured nature and are easy to implement compared to the structured pruning methods. Further, FLOPs of two methods can be exactly the same, while their performance scores may differ due to the presence of multiple layers with similar FLOPs. However, if any one of these binary layers is interchanged from the set of layers having equal FLOPs, it may negatively impact B2NNs' performance.

Table 2. Performance scores for SiamFC with ResNet-18 backbone on GOT-10K datasets for four compression methods: Network Slimming, Adaptive MBQ, BNAS and MixBin (ours). The Full precision network has a total of 4.28×10^8 FLOPs, while the binary network generated using the Bi-RealNet method 2.00×10^7 FLOPs. The term "budget" refers to the percentage of FLOPs that remain after applying the respective compression method.

Method	Budget(%)	AO↑	$SR_{0.5}$↑	FLOPs(%)↓
Full Precision Network	-	0.251	0.242	100
Binary Network	-	0.189	0.173	4.68
Network Slimming	60	0.235	0.220	60
Adaptive MBQ		0.215	0.198	56.67
BNAS		0.211	0.186	61.01
MixBin$_{Loss}$		**0.237**	**0.219**	**56.57**
MixBin$_{Grad}$		**0.240**	**0.234**	**61.01**
Network Slimming	40	0.198	0.179	40
Adaptive MBQ		0.210	0.199	39.34
BNAS		0.211	0.199	39.34
MixBin$_{Loss}$		**0.235**	**0.226**	**43.67**
MixBin$_{Grad}$		**0.236**	**0.228**	**43.67**
Network Slimming	10	0.201	0.182	10
Adaptive MBQ		0.218	0.202	13.35
BNAS		0.218	0.202	13.35
MixBin$_{Loss}$		**0.226**	**0.216**	**13.35**
MixBin$_{Grad}$		**0.226**	**0.216**	**13.35**

Building light weight object trackers. Object tracking is an application domain that benefits among the most from model compression. When deployed on low-power devices, object trackers need to be light-weight and deliver desired inference speed based on the target hardware. In this regard, we demonstrate the application of MixBin to design light-weight object trackers. We analyze the stability of the compressed variants of the ResNet-18 model at various FLOPs budgets and analyze how well it fairs against model compression achieved using other baselines.

Table 2 shows the results for various compressed variants of ResNet-18. From Table 2, we see that the compressed models obtained from MixBin$_{Grad}$ as well MixBin$_{Loss}$ are superior than any other choice of compressions. While this difference is small at FLOPs budget of 60% of the original network, it grows to a large margin at extreme levels of preserving only 10% FLOPs from the original network, further confirming the applicability and effectiveness of MixBin. Note that in some cases two methods gave exactly the same set of layers which led to similar scores.

Table 3. Performance scores of B2NN masks transferred (Trans.) from the ImageNet-1K dataset to both a classification task on the TinyImageNet [TinyIN] and CIFAR-100 [C100] dataset and an object tracking task on the GOT-10K dataset, using the ResNet-18 architecture. The term "budget" refers to the percentage of FLOPs that remain after applying the respective compression method. The performance metrics evaluated for the classification task are accuracy (Acc.), while the metrics evaluated for the tracking task are average overlap (AO) and success rate at 0.5 overlap ($SR_{0.5}$)

		Classification		Object Tracking	
Method	Budget (%)	TinyIN ↑	C100 ↑	AO ↑	$SR_{0.5}$ ↑
Trans.	60	56.73	56.27	0.232	0.217
Base		**57.11**	**56.37**	**0.240**	**0.234**
Trans.	40	55.67	**55.91**	**0.237**	**0.232**
Base		**56.44**	55.48	0.236	0.228
Trans.	20	**55.51**	**54.69**	**0.220**	**0.206**
Base		55.08	52.93	0.214	0.203
Trans.	10	**54.46**	**53.05**	0.221	0.211
Base		54.25	52.45	**0.226**	**0.216**

Transferability of B2NNs models. The generalizibility of the B2NNs obtained from MixBin can be assessed based on the extent to which they are transferable across datasets. This implies analyzing how well a model compressed on one dataset performs on another dataset. In this regard, we present results for the scenario of model transfer from ImageNet-1K to TinyImageNet and CIFAR-100 dataset for classification and from ImageNet-1K to GOT-10K for object tracking. These are reported in Table 3. From the results, we see that the performance of the transferred models is in a similar range as when the model is trained on the base dataset. Clearly, B2NNs designed for large datasets such as ImageNet seem to work well for the base datasets, thus these do not require any additional layer selections. An interesting observation made on the classification tasks is that at budget of as low as 20% of the original FLOPs, the transferred B2NNs seems to perform slightly better than those trained on the base dataset. Overall, B2NNs obtained from MixBin seem to generalize well across datasets.

5. Conclusion and Future work

In this paper, we have proposed a strategy to perform partial binarization of neural networks in a controlled sense, thereby constructing *budgeted binary neural network (B2NN)*. We presented MixBin, an efficient strategy for finding a well optimized mixture of binary and full precision components in a given network architecture. MixBin allows to explicitly choose the approximate fraction of the network to be kept as binary, thereby presenting the flexibility to adapt the inference cost to a prescribed budget. Numerical experiments conducted on various datasets and model choices support our claim that the B2NNs obtained from our MixBin strategy are significantly better than those obtained from random selection, iterative selection, and even more elegant strategies based on popular methods from the field of model compression. This is strongly evident from the results presented on ImageNet-1K dataset as well as the downstream task of object tracking where significant improvements over the baselines are reported. Overall, from the results and discussions presented earlier in the paper, it can be concluded that MixBin is an efficient and effective strategy for constructing B2NNs that can maximally utilize the available computational resources.

Limitations and future work. Although the presented MixBin strategy results in mixing of full-precision and binary components which are superior over the chosen competitive baselines at similar budgets, there is still scope of improving it. In this paper, we have focused on optimizing the selection process at the layer level. However, there is potential for further investigation into how the network's behavior will be affected by a more granular approach, specifically at the channel level. Additionally, it would be intriguing to explore the application of our layer selection method for transformer-based architectures and assess its efficiency in that context. Moreover, the currently available hardware are limited in terms of fully exploiting the power of binary neural networks, and similar limitation exists for the B2NNs designed using our methodology. However, we believe that with the rapid developments happening in this field, this should be resolved soon.

References

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1

[2] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *International Conference on Computer Vision (ICCV)*, 2017. 1, 3, 6

[3] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015. 1, 3

[4] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *ArXiv*, abs/1806.08342, 2018. 1, 3

[5] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv: Learning*, 2016. 1, 3

[6] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2016. 2, 3

[7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015. 2

[8] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *ArXiv*, abs/1711.11294, 2017. 2, 3

[9] Brais Martínez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. *ArXiv*, abs/2003.11535, 2020. 2

[10] Zechun Liu, Wenhan Luo, Baoyuan Wu, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Binarizing deep network towards real-network performance. *International Journal of Computer Vision*, 128(1):202–219, 2020. 2, 3, 4, 6

[11] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv: Learning*, 2019. 2, 3

[12] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 3

[13] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *ArXiv*, abs/1602.07360, 2016. 3

[14] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017. 3

[15] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578, 2017. 3

[16] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, 2018. 3

[17] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2815–2823, 2019. 3

[18] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Özlem Aslan, Shengjie Wang, Abdel rahman Mohamed, Matthai Philipose, Matthew Richardson, and Rich Caruana. Do deep convolutional nets really need to be deep and convolutional? *arXiv: Machine Learning*, 2017. 3

[19] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014. 3

[20] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2015. 3

[21] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive representation distillation. *ArXiv*, abs/1910.10699, 2020. 3

[22] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2017. 3

[23] Yihui He, X. Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, 2017. 3

[24] Xin Dong, Shangyu Chen, and Sinno Jialin Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 3

[25] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity. *ArXiv*, abs/1810.02340, 2019. 3

[26] C. Lemaire, A. Achkar, and P. Jodoin. Structured pruning of neural networks with budget-aware regularization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9108–9116, 2019. 3

[27] Rishabh Tiwari, Udbhav Bamba, Arnav Chavan, and Deepak Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations. In *International Conference on Learning Representations (ICLR)*, 2021. 3

[28] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018. 3

[29] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems*, 33:6377–6389, 2020. 3

[30] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Aciq: Analytical clipping for integer quantization of neural networks. *ArXiv*, abs/1810.05723, 2018. 3

[31] Xiandong Zhao, Ying Wang, Xuyi Cai, Chuanming Liu, and Lei Zhang. Linear symmetric quantization of neural networks for low-precision integer hardware. In *International Conference on Learning Representations (ICLR)*, 2020. 3

[32] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *ArXiv*, abs/2103.13630, 2022. 3

[33] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. *ArXiv*, abs/2003.03488, 2020. 3, 4

[34] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2247–2256, 2020. 3

[35] Chunlei Liu, Peng Chen, Bohan Zhuang, Chunhua Shen, Baochang Zhang, and Wenrui Ding. Sa-bnn: State-aware binary neural network. In *AAAI Conference on Artificial Intelligence*, 2021. 3

[36] Ameya Prabhu, Vishal Batchu, Rohit Gajawada, Sri Aurobindo Munagala, and Anoop Namboodiri. Hybrid binary networks: Optimizing for accuracy, efficiency and memory. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 821–829, 2018. 3

[37] Hanlin Chen, Li'an Zhuo, Baochang Zhang, Xiawu Zheng, Jianzhuang Liu, David S. Doermann, and Rongrong Ji. Binarized neural architecture search. In *AAAI Conference on Artificial Intelligence*, 2019. 3, 6

[38] Sijie Zhao, Tao Yue, and Xuemei Hu. Distribution-aware adaptive multi-bit quantization. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9277–9286, 2021. 3, 6

[39] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18518–18529. Curran Associates, Inc., 2020. 3, 6

[40] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763, 2017. 6