MixBin: Towards Budgeted Binarization

Anonymous authors
Paper under double-blind review

Abstract

Binarization has proven to be amongst the most effective ways of neural network compression, reducing the FLOPs of the original model by a large extent. However, such levels of compression are often accompanied by a significant drop in the performance of the model. There exist some approaches that reduce this performance drop by facilitating partial binarization of the network, however, a systematic approach to mix binary and full-precision parameters in a single network is still missing. 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)*. We present `MixBin`, an iterative search-based strategy that constructs B2NN through optimized mixing of the binary and full-precision components. `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 at a prescribed budget. We demonstrate through numerical experiments that B2NNs obtained from our `MixBin` strategy are significantly better than those obtained from random selection of the network layers. To perform partial binarization in an effective manner, it is important that both the full-precision as well as the binary components of the B2NN are appropriately optimized. We also demonstrate that the choice of the activation function can have a significant effect on this process, and to circumvent this issue, we present BinReLU, an integral component of `MixBin`, that can be used as an effective activation function for the full-precision as well as the binary components of any B2NN. Experimental investigations reveal that BinReLU outperforms the other activation functions in all possible scenarios of B2NN: zero-, partial- as well as full binarization. Finally, we demonstrate the efficacy of `MixBin` on the tasks of classification and object tracking using benchmark datasets.

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 ResNet18 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 (Russakovsky et al., 2015). 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 dropping the non-important set of parameters or channels (pruning) (Liu et al., 2017), distilling knowledge of the dense teacher network into a light-weight student network (distillation) (Hinton et al., 2015), converting 32-bit representations of the parameters to half-precision or even lower (quantization) (Krishnamoorthi, 2018), and converting all the parameters of network to 1-bit representations (binarization) (Courbariaux et al., 2016).

Among the methods outlined above, binarization is very 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 (Rastegari et al., 2016). However, due to the significantly reduced representation, the performance of the binarized network is significantly lower than its dense counterpart. To circumvent this, several approaches exist such as binarizing only the weights and keeping the activations as full-precision (Courbariaux et al., 2015), parallely stacking multiple binarized layers (Lin et al., 2017), using special layers such as squeeze-and-excitation blocks (Martínez et al., 2020) and retaining skip connections as full-precision modules in a ResNet-type binary network (Liu et al., 2020a). All the above methods exploit partial binarization of the network such that the extent of performance drop due to binarization is low. 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 compressed model and the drop in 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.

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. 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 a later section of this paper.

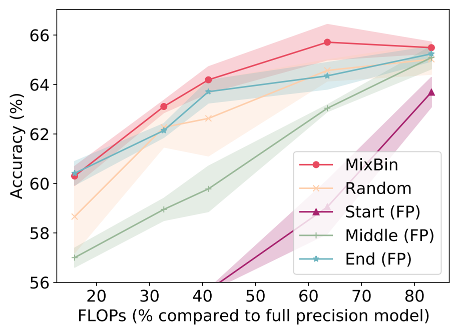


Figure 1: Performance scores obtained on CIFAR100 dataset by different compressed variants of cResNet20 model. The compressed variants are obtained by mixing full-precision (FP) and binary layers using different strategies: random, keeping full-precision in the start, middle and end of the network with the rest of it as binary, and with our `MixBin` approach.

To overcome the issue outlined above, we present `MixBin`, an iterative search 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 (referred further as *budgeted binarization*). We demonstrate through numerical experiments that B2NNs obtained from our `MixBin` strategy are significantly better than those obtained from random selection of the network layers. Our `MixBin` is stable with respect to the employed initialization scheme, and we demonstrate that when following different trajectories for iterative selection of the layers to binarize, the results obtained for different choices of the FLOP budget are approximately similar. Beyond this, we demonstrate that `MixBin` is path independent. This implies that for any prescribed budget, the performance of the final B2NN obtained from iterative binarization of parts of a full-precision model (`MixBin`-shrink) is approximately similar to that obtained when a fully binary network is iteratively converted into a B2NN (`MixBin`-grow). Lastly, `MixBin` leads to B2NNs which are transferable across datasets, implying that a compressed model designed on one dataset fairs well on other datasets as well.

To perform partial binarization in an effective manner, it is important that both the components of the B2NN are appropriately optimized. For a full-precision network, it is common to use ReLU activations, however, since it discards values below 0, this activation function does not work well for BNNs. On the other hand, it is common to use HardTanh or identity as activation function for BNNs (Qin et al., 2020; Liu et al., 2020a). Through experiments, we demonstrate later in this paper, that these activations are also not suited for B2NNs. Clearly an activation function suited for B2NN construction is still missing in the existing literature. To circumvent this issue, we present BinReLU, an integral component of `MixBin`, that can be used as an effective activation function for the full-precision as well as the binary components of any B2NN. Experimental investigations reveal that BinReLU outperforms the other activation functions in all possible scenarios of B2NN: zero-, partial- as well as full binarization.

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

- We introduce budgeted binary neural networks (B2NNs), compressed variants of the dense full-precision models obtained through partial binarization. B2NN relies on identifying the right set of layers that are to be retained as binary or full-precision.
- We present `MixBin`, an iterative search strategy to compress a model 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. Further, the resultant B2NNs are transferable in nature, thereby when trained on one dataset, these work well on other datasets as well.
- We demonstrate that `MixBin` can be used to 'grow' a binary network through converting part of it to full-precision as well as 'shrink' a real-valued network through its partial binarization. For a given choice of FLOPs, we demonstrate that both the above routes result in models with almost similar performance.
- Mixing full-precision and binary layers requires that both the components are properly optimized in a single framework, and to facilitate this, we present BinReLU activation function, an integral part of `MixBin`, which works well for the B2NN frameworks.
- To demonstrate the efficacy of `MixBin`, we conduct experiments on various classification models for the popular benchmark datasets. We further demonstrate that when used for object tracking, `MixBin` results in light-weight yet very efficient trackers that are almost optimally compressed.

2 Related Work

Neural networks are known to be overparameterized for most of the tasks that they are used for (Frankle & Carbin, 2019). This overparameterization causes increased FLOPs with little to no effect on the model accuracy, and creates undesired additional latency when deploying these models in production. There exist several work that focus on addressing these issues by making neural networks models more efficient.

First among these is through inducing efficient components in the neural network design, such as using residual networks with bottleneck blocks (He et al., 2016), SqueezeNets, which replace some of the 3×3 convolutions with 1×1 (Iandola et al., 2016), using depthwise separable convolutions as in MobileNet (Howard et al., 2017) and neural architecture search (Zoph & Le, 2017; Pham et al., 2018; Tan et al., 2019). Another approach to design efficient networks involves distilling the information of large networks into smaller networks (knowledge distillation) (Hinton et al., 2015; Urban et al., 2017; Ba & Caruana, 2014; Romero et al., 2015; Tian et al., 2020). Further, there exist works that aim at compressing the original network itself through identifying the undesired or less desired weights or filters of a network and removing them. Examples of pruning include removing weights/connections through ranking scores calculated via L1/L2 norms of parameters (Li et al., 2017; He et al., 2017; Frankle & Carbin, 2019), identifying the channels to remove based on the activations resulting from the respective layer Luo et al. (2017), using derivatives to identify the parts to remove (Dong et al., 2017; Lee et al., 2019) and learning masks that represent the parts to retain or remove (Lemaire et al.; Tiwari et al., 2021; Liu et al., 2017). A last but 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 (Krishnamoorthi, 2018; Banner et al., 2018; Zhao et al., 2020; Gholami et al., 2022).

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 (Courbariaux et al., 2016). Due to the replacement of the conventional convolution operation with a bitwise operation, a significant computational gain is observed. Further, adding the channelwise scaling the the binary weights allows to scale BNNs to largescale datasets such as ImageNet (Rastegari et al., 2016).

Binarization is a very effective method for compressing the networks and making them efficient, however, due to significant reduced representation of the network, the performance is significantly reduced. There exist several works that attempt to find a right balance between the drop in 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 together to increase the representation capability of the network (Lin et al., 2017). In BiRealNet (Liu et al., 2020a), 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 (Liu et al., 2020b), IR-Net (Qin et al., 2020) and SA-BNN (Liu et al., 2021).

Most of the approaches listed above attempt to find representations in between a fully binary network and a fully 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 that performs selective binarization through locally converting the activations to full-precision and retaining the rest of the network as binary (Prabhu et al., 2018). However, this approach also provides limited flexibility in terms of full exploitation of the mixing between binary and full-precision components. Our `MixBin` strategy effectively mixes the binary and full-representations for weights as well as activations to build B2NNs that are well optimized.

3 MIXBIN

3.1 BACKGROUND

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

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

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 $32\times$ 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 Rastegari et al. (2016).

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, 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 $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n, \mathbf{w}_{n+1}\}$ and $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n\}$. 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 to retain the input, output and $\mathbf{w}_{n+1}$ as full-precision. For methods that perform complete binarization, $\mathbf{w}_i \ \forall \ i \in [1, n]$ and $\mathbf{a}_i \ \forall \ i \in [1, n]$ 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 $\mathbf{a}_i$ and $\mathbf{w}_i$ together as $\boldsymbol{\theta}_i = \{\mathbf{a}_i, \mathbf{w}_i\}$ and performs binarization on a subset $\boldsymbol{\Phi} \subset \boldsymbol{\Theta}$, where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_n\}$. During The generic mathematical problem that we solve with B2NN can be stated as follows.

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

where $\boldsymbol{\Phi}^*$ 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 $\mathcal{B}_0$ refers to 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 on the choice of $\boldsymbol{\Phi}$. A simple and straightforward approach to construct B2NN is to randomly sample $\boldsymbol{\Phi}$ from the set of layers defined by $\boldsymbol{\Theta}$. While the resultant B2NN is compressed in the desired manner, the performance of the resultant model can be significantly deteriorated, and this is demonstrated in Figure 1. We further demonstrate that binarizing layers from different parts of the network can affect the performance of the resultant B2NN differently. As shown in Figure 1, keeping more full-precision layers towards the end of the network favors its performance. And binarizing the later layers affects the performance of the resultant B2NN adversely. Clearly, mixing full-precision and binary layers randomly is not the right choice, and this issue is tackled by the proposed `MixBin` strategy.

MixBin. It is an iterative search strategy designed to identify the right layers of any given network to be converted to binary or full precision, one-by-one. The approach of `MixBin` is described in Algorithm 1. For the k out of n layers to be binarized, the jth step of binarization, where $j \in [1, k]$, can be stated as finding the optimal layer $\boldsymbol{\theta}^* \in \boldsymbol{\Theta}_{(j)}$ to be binarized. It can be mathematically stated as follows.

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

where $\boldsymbol{\Theta}_{(j)} = \boldsymbol{\Theta} - \boldsymbol{\Phi}_{(j)}$. Here, $\boldsymbol{\Phi}_{(j)}$ denotes the layers that have already been binarized in the previous $j - 1$ steps and is defined as $\boldsymbol{\Phi}_{(j)} = \{\boldsymbol{\theta}^*_{(1)}, \boldsymbol{\theta}^*_{(2)}, \dots, \boldsymbol{\theta}^*_{(j-1)}\}$, where $\boldsymbol{\theta}^*_j$ denotes the optimal layer chosen at the jth step of binarization to obtain B2NN. For calculating $\boldsymbol{\theta}^*_j$, we perform brute search over all elements of $\boldsymbol{\Theta}_{(j)}$ and choose the layer, which when binarized, maximizes the performance of the intermediate B2NN model.

Algorithm 1: `MixBin` (Shrink) Approach

```
Given  : Empty set $\{\}$;
         Current layer chosen $\boldsymbol{\theta}_j$;
         Optimal layer chosen $\boldsymbol{\theta}^*_j$;
         Current objective function $\mathcal{L}$;
         Optimal objective function $\mathcal{L}^*$;
Input  : Network weight set $\boldsymbol{\Theta}$
Output: Binarized weight set $\boldsymbol{\Phi}$
$\boldsymbol{\Phi} \leftarrow \{\}$
for $j = 1 \dots k$ do
    $\mathcal{L}^* \leftarrow \infty$
    for $\boldsymbol{\theta}_j \in \boldsymbol{\Theta}$ do
        $\mathcal{L} \leftarrow \text{MIXBIN}(\boldsymbol{\Phi}, \boldsymbol{\Theta} - \boldsymbol{\theta}_j, \boldsymbol{\theta}_j)$
        if $\mathcal{L} < \mathcal{L}^*$ then
            $\mathcal{L}^* \leftarrow \mathcal{L}$
            $\boldsymbol{\theta}^*_j \leftarrow \boldsymbol{\theta}_j$
        end
    end
    $\boldsymbol{\Phi} \leftarrow \text{PUSH}(\boldsymbol{\theta}^*_j)$
    $\boldsymbol{\Theta} \rightarrow \text{POP}(\boldsymbol{\theta}^*_j)$
end
```

The approach described above states how `MixBin` iteratively converts a full-precision model into a binary one. Alternatively, `MixBin` can also be used in a reverse order to convert a binary neural network into a B2NN. The underlying strategy is still the same, and it involves converting the layers of a binary model into full-precision, in an iterative manner.

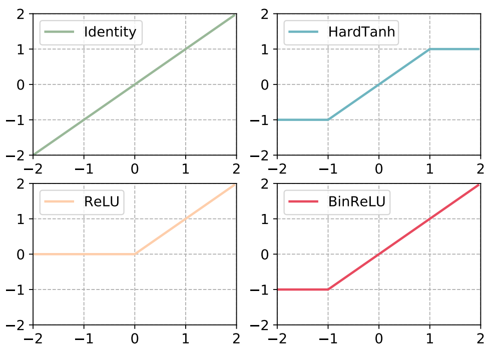


Figure 2: Schematic representation of BinReLU and other activation functions for an input range of [-2, 2].

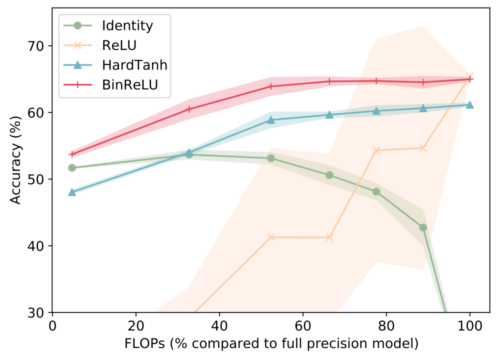


Figure 3: Performance of B2NNs with different activation functions obtained at various FLOPs on CIFAR100 with cResNet20 model.

3.3 BinReLU activation function

The proposed BinReLU activation function is designed to enhance the stability of the full-precision as well as binary components of a B2NN in general. Mathematically, BinReLU can be stated as

$$f(x) = \begin{cases} -1 \text{ if } x \leq -1 \\ x \text{ otherwise} \end{cases}, \tag{6}$$

where x and $f(x)$ denote the input and output of the activation function.

Figure 2 shows the BinReLU function together with ReLU, HardTanh and Identify mappings. For full-precision networks, ReLU is considered a very effective choice of activation, however, since it eliminates the activation information below 0, it does not work well for binary networks. For BNNs, either of HardTanh or Identity functions are preferred. However, both these activations do not work well for the real-valued networks (Figure 3). Note that an identity mapping works well for BNN since for such cases, nonlinearity is inherently introduced through the squashing of the activation values to -1 and 1 using Sign$(\cdot)$ as denoted in Eq. 1. BinReLU is inspired from the other activations stated here in a sense that it preserves the characteristics of ReLU for positive activations and keeps them real-valued, and also ensures that the activation information between -1 and 0 is preserved.

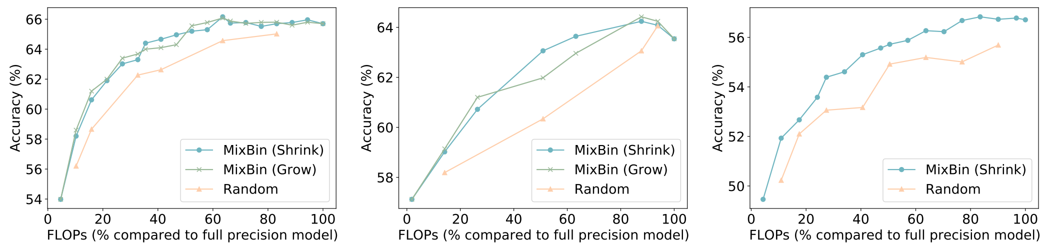
4 Experiments

We have conducted multiple experiments to demonstrate the efficacy of the `MixBin` approach as well as the BinReLU activation function. First we describe the setup for the presented experiments followed by results and insights for each of them.

Experimental setup. For the experiments presented in this paper, we consider the tasks of image classification and object tracking. For image classification, we conduct experiments to compress CIFAR-ResNet20 (referred further as cResNet20), ResNet18 and VGG11 models on CIFAR100 and as well as compress ResNet18 on TinyImageNet (TinyIM) dataset. For the task of object tracking, we consider the popular SiamFC model (Bertinetto et al., 2016), and train and compress the tracker on GOT-10K train set. The performance of the trackers resulting from `MixBin` is evaluated on the GOT-10K test dataset. For details related to the hyperparameter configurations, see Appendix A.1.

4.1 Choice of activation function

We study here the effect of different activations including BinReLU on the performance of B2NNs obtained at different choices of the FLOP budget. We compare our results with ReLU, Identity and HardTanh functions since these are the popular choices for full-precision and BNN models, respectively. Figure 3 shows the performance scores for the different activation functions obtained for the classification of samples from the CIFAR100 dataset using cResNet20 model.



(a) cResNet20 / CIFAR100 (b) VGG 11 / CIFAR100 (c) ResNet18 / TinyImageNet

Figure 4: Performance scores obtained at different compression levels for three different model and data combinations.

We observe that BinReLU outperforms the other activation functions for all the constructions of B2NN and levels performance with ReLU for the fully real-valued network. While ReLU seems to produce good results for the full-precision model, it fails severely for almost all constructions of B2NNs. For compressed models with FLOPs reduced to around 40% of the original model, the performance of the model drops by approximately 50%. Identity mapping seems to work at very low FLOPs, but as expected the performance dips when the model moves more towards a full precision one, the reason accounting to the fact that the extent of nonlinearity gets reduced as more full-precision components are added. Further, as observed HardTanh seems to be the second most reliable choice, however, the performance obtained with this activation is significantly lower than BinReLU at all levels of compression. Clearly, BinReLU seems to be a well suited choice for building B2NNs.

4.2 MixBin: B2NN WITH ITERATIVE SEARCH

We experimentally study how the compressed models obtained using `MixBin` fair against those where the layers to binarize are chosen randomly. Further, we conduct additional experiments to provide deeper insights into `MixBin`. In this regard, we conduct multiple experiments and the results are stated below.

Iterative identification of target layers. First we investigate how iteratively applying `MixBin` leads to well compressed models. We base our comparison in terms of measuring the classification accuracy at the same level of FLOPs. Note that FLOPs reported here refer to sum of the FLOPs from the full-precision part of the network and BOPs from binary parts (see details in Appendix B). We compare the results of `MixBin` with B2NNs constructed from random selection of binary layers, and the obtained results are presented in Figure 4.

From the Figure 4, we see that `MixBin` outperforms the results of random selection by large margins at most of the FLOP budgets. This backs the efficiency claim of `MixBin` in designing efficient B2NNs for any given choice of FLOP budgets. An interesting insight is that for the initial levels of compression, the performance of the model is quite stable with almost no significant drop in the performance of the model. For some cases we see that the performance increases slightly and then starts dipping again when the model is further compressed. The reason for initial increase can be associated with reduced overparameterization and better generalization of the resultant B2NN on the test dataset. At larger compression levels, the performance of the B2NNs starts dipping faster denoting that the representation capability of the network is relatively insufficient for the target dataset.

Invariance to the direction of MixBin construction. As described earlier, `MixBin` can be operated in two different modes, grow and shrink, referring to conversions from binary to full precision and full precision to binary, respectively. Figure 4 shows the results for both the modes. It is evident from the plots that `MixBin` is approximately invariant to the direction of B2NN construction, implying that for any given FLOP budget, approximately similar model performance can be obtained by binarizing full precision components or converting binary parts to real-valued.

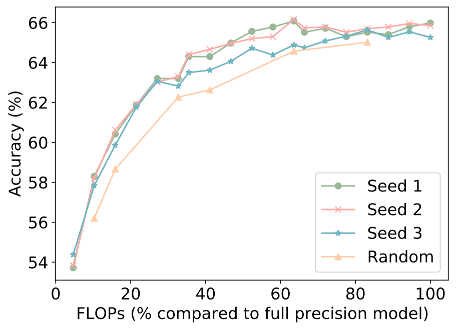


Figure 5: Performance scores obtained with different random seeds for with MixBin for the various compression levels of cResnet20 on CIFAR100 dataset.

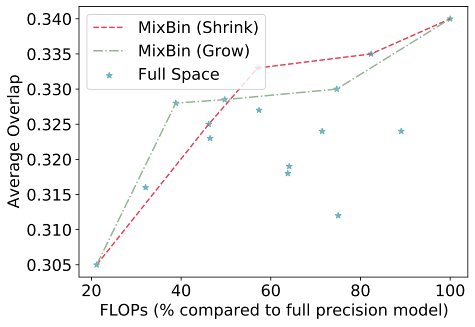


Figure 6: Performance scores on GOT-10K for SiamFC and its compressed variants obtained using B2NNs with different combinations of full-precision and binary layers.

Table 1: Demostrating the transferability of MixBin compressed models trained on TinyImageNet and transferred to CIFAR100 dataset. FLOPs for full precision network is 1.4×10^8.

Remaining FLOPs (%)	MixBin (Shrink)	Random	Transfer from TinyIN
50.94	68.20	63.66	67.01
37.32	68.02	66.24	67.44
20.82	66.62	63.30	66.21
14.22	65.86	63.28	65.86

Effect of random seed. The resultant models obtained from MixBin are stable with respect to the initialization seed used to train the models. We confirm this through running MixBin with three different random seeds and related results are presented in Figure 5. We observe that the performance scores for the 3 runs do not differ by significant margins at all compression levels. Further, all the 3 cases consistently outperform the model with random selection of the binary layers. These observations confirm that the compression strategy of MixBin is not sensitive to the choice of initialization.

4.3 Transferability of MixBin models

The generaliziblity of the compressed models 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 dataset performs on another dataset. In this regard, we present results for the scenario of model transfer from TinyIN to CIFAR100 dataset and the results are reported in Table 1. From the results, we see that the performance of the transferred models is approximately similar to those obtained directly on CIFAR100 on the respective budgets. Moreover the scores of the transferred models are consistently superior over the random selection method. These results clearly demonstrate the transferability of the B2NNs obtained from MixBin.

4.4 Building light-weight budgeted 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 SiamFC model at various FLOPs budgets and analyze how well it fairs against random compression.

Figure 6 shows the results for various compressed variants of SiamFC. Each data point on the plot corresponds to one or more layers of the SiamFC backbone converted from full-precision to binary. SiamFC model uses AlexNet backbone with 5 convolutional layers and different combinations are experimented where some of the layers are kept as full-precision and the rest as binary. Some example combinations are ‘FBBBB’, ‘FBFBF’ and ‘FBBBB’, among others. Note that the first layer is always retained as full-precision. From Figure 6, we see that the the compressed models obtained from `MixBin` in shrink mode as well as grow mode are almost always better than any other choice of compressions, thereby confirming that the results of `MixBin` at intermediate budgets are generally close to optimal for the SiamFC object tracking problem. This clearly demonstrates that `MixBin` is an effective method to design compressed models in an optimized sense for tasks such as object tracking and possibly object detection, such that the performance on these tasks is maximized.

5 Conclusions 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 iterative search-based strategy that constructs B2NN through optimized mixing of the binary and full-precision components. `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 of the network layers. To perform partial binarization in an effective manner, we have also presented BinReLU, an integral component of `MixBin`, that can be used as an effective activation function for the full-precision as well as the binary components of any B2NN. Experimental investigations reveal that BinReLU outperforms the other activation functions in all possible scenarios of B2NN: zero-, partial- as well as full binarization.

Limitations and future work. The presented `MixBin` strategy is the first step towards mixing full-precision and binary layers together in an effective manner, and it performs significantly better than the random choice. However, we believe that this is still not the optimal solution, and a more principled strategy can be developed with further research on this aspect.

Reproducibility Statement

We describe here details which are important for the reproducibility of the results presented in this paper. For all experiments, the associated hyperparameter details as well as the hardware choice are described in Appendix A. All the experiments were run on three different seeds to circumvent the effect of randomness. For the sake of completeness, we will also released our code which was used to run all the experiments.

References

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

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

Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision Workshops (ECCVW)*, 2016.

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.

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.

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.

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

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.

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

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

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

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.

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.

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

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

C. Lemaire, A. Achkar, and P. Jodoin. Structured pruning of neural networks with budget-aware regularization.

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

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

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.

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, 2020a.

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

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.

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

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

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.

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)*, pp. 821–829, 2018.

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)*, pp. 2247–2256, 2020.

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.

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.

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.

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)*, pp. 2815–2823, 2019.

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

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.

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.

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.

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

APPENDIX

A TRAINING DETAILS

A.1 HYPER-PARAMETERS

VGG-11, ResNet-20, ResNet-18 were trained with batch size of 128 at initial learning rate of 0.2 using SGD optimizer with momentum 0.9 and weight decay 10^{-3}. Cross Entropy Loss was used with label smoothing of 0.01. We use step learning rate strategy to decay learning rate by 0.1 after every 50% and 75% of total epochs. For CIFAR100, models were trained for 160 epochs with RandomCrop and RandomHorizontalFlip augmentations whereas for TinyImageNet the number of epochs were reduced to 120 and, RandomAffine and RandomHorizontalFlip was used as augmentations. . **SiamFC** was trained with a batch size of 8 at an initial learning rate of 10^{-2} using SGD optimizer with momentum 0.9 and weight decay 5×10^{-4}. We use Expontial learning rate schedular with gamma value of 0.59 and final learning rate of 10^{-5}. All experiments were trained for 50 epochs. Please refer to open-source implementation of `https://github.com/huanglianghua/siamfc-pytorch` for further details.

A.2 HARDWARE

All experiments were run on Nvidia V100 32GB card with 512 GB RAM and 64 core processor.

B 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 as well as the binary counterpart. FLOPs for binary counterpart are calculated based on the convention presented in Liu et al. (2020a), i.e., to divide the full-precision FLOPs by 64.

C EXPERIMENTS: ADDITIONAL DETAILS

Table 2: Performance of B2NNs with different activation function obtained at various FLOPs on CIFAR100 with cResNet20. FLOPs for full precision network is 4.14×10^7.

	Activation function			
Remaining FLOPs (%)	Identity	ReLU	HardTanh	BinReLU
100.00	16.40	65.44	61.15	64.98
88.78	42.72	54.66	60.64	64.53
77.57	48.12	54.33	60.23	64.72
66.35	50.60	41.23	59.65	64.66
52.33	53.13	41.28	58.87	63.90
32.71	53.68	29.18	54.00	60.48
4.67	51.69	20.39	48.05	53.72

Table 3: Performance scores obtained for MixBin at various FLOPs on CIFAR100 with VGG11. FLOPs for full precision network is 1.51×10^8.

Remaining FLOPs (%)	MixBin (Shrink)	MixBin (Grow)	Random
1.88	57.12	56.7	–
14.14	59.02	59.14	58.19
26.41	60.72	61.2	–
50.94	63.06	61.98	60.34
63.20	63.64	62.96	–
87.73	64.24	64.42	63.06
93.87	64.08	64.24	64.06
100.00	63.54	63.42	–

Table 4: Performance scores obtained for MixBin at various FLOPs on CIFAR100 with cResNet20. FLOPs for full precision network is 4.14×10^7.

Remaining FLOPs (%)	MixBin (Shrink)	MixBin (Grow)	Random
4.67	53.98	53.98	–
10.28	58.2	58.3	–
15.88	60.62	59.42	58.66
21.49	61.9	62	–
27.10	63.02	63.4	–
32.71	63.3	63.68	62.27
35.51	64.4	64	–
41.12	64.66	64.98	62.63
46.73	64.96	64.3	–
52.33	65.2	65.56	–
57.94	65.3	65.78	–
63.55	66.16	66.08	64.57
66.35	65.74	65.52	–
71.96	65.78	65.72	–
77.57	65.52	65.2	–
83.18	65.7	65.8	65.02
88.78	65.78	65	–
94.39	65.96	65.8	–
100.00	65.70	65.70	–

Table 5: Performance scores obtained for MixBin at various FLOPs on TinyImageNet with ResNet18. FLOPs for full precision network is 5.63×10^8.

Remaining FLOPs (%)	MixBin (Shrink)	Random
4.32	49.46	–
10.92	51.93	50.23
17.52	52.67	52.1
24.12	53.58	–
27.42	54.39	53.06
34.02	54.61	–
40.62	55.3	53.17
47.21	55.57	–
50.51	55.72	54.92
57.11	55.88	–
63.71	56.27	55.19
70.31	56.23	–
76.91	56.68	55.01
83.50	56.83	–
90.10	56.73	55.69
96.70	56.78	–
100.00	56.71	–

Table 6: Performance of SiamFC and its different compressed variants obtained using B2NNs with different combinations of full-precision and binary layers. Here, method refers to which layers of SiamFC are binarized where F stands for full-precision and B stand for binarization.

Method	Average Overlap	Remaining FLOPs (%)
FBBBB	0.305	21.17
FBBBF	0.316	32.08
FBBFB	0.328	38.82
FFBBB	0.325	46.15
FBFBB	0.323	46.47
FBBFF	0.329	49.72
FBFFB	0.319	64.11
FFFBB	0.324	71.45
FBFBF	0.327	57.38
FFBBF	0.333	57.06
FFBFB	0.318	63.80
FBFFF	0.312	75.02
FFBFF	0.330	74.70
FFFBF	0.335	82.36
FFFFB	0.324	89.09
FFFFF	0.340	100.00