Training Competitive Binary Neural Networks from Scratch

Joseph Bethge*, Marvin Bornstein†, Adrian Loy†, Haojin Yang*, Christoph Meinel*
Hasso Plattner Institute, University of Potsdam, Germany
P.O. Box 900460, Potsdam D-14480
*firstname.surname@hpi.de, †firstname.surname@student.hpi.de

Abstract

Convolutional neural networks have achieved astonishing results in different application areas. Various methods that allow us to use these models on mobile and embedded devices have been proposed. Especially binary neural networks are a promising approach for devices with low computational power. However, training accurate binary models from scratch remains a challenge. Previous work often uses prior knowledge from full-precision models and complex training strategies. In our work, we focus on increasing the performance of binary neural networks without such prior knowledge and a much simpler training strategy. In our experiments we show that we are able to achieve state-of-the-art results on standard benchmark datasets. Further, to the best of our knowledge, we are the first to successfully adopt a network architecture with dense connections for binary networks, which lets us improve the state-of-the-art even further. Our source code can be found online: https://github.com/hpi-xnor/BMXNet-v2

1. Introduction

Nowadays, significant progress through research is made towards automating different tasks of our everyday lives. From vacuum robots in our homes to entire production facilities run by robots, many tasks in our world are already highly automated. Other advances, such as self-driving cars, are currently being developed and depend on strong machine learning solutions. The amount of apps on smartphones, which adopt deep learning techniques to solve a variety of tasks, is rising rapidly and will likely continue to do so in the future. All these devices have limited computational power, often while trying to minimize energy consumption, but have many use cases for machine learning.

We will consider the example of a fully automated self-driving car. It is crucial for such a system to achieve high accuracy coupled with guaranteed real-time image processing. Furthermore, the image processing system needs to be hosted in the car itself, as a stable Internet connection with low latency cannot be guaranteed in this setting. This requirement limits the available computational power and memory, but at the same time profits from a low energy consumption. A promising technique that can deal well with these conditions are Binary Neural Networks (BNNs). In a BNN the commonly used full-precision weights of a convolutional neural network are replaced with binary weights. This results in a storage compression by a factor of 32× and allows for significantly more efficient inference on CPU-only architectures.

We discuss existing approaches in Section 2. Moreover, we identified three ways to increase the accuracy of a binary model and describe how we applied them to a binary network with dense shortcut connections: removing bottleneck designs, increasing the number of shortcut connections throughout the network, and replacing certain layers with full-precision layers. We describe these and other common techniques together with our implementation details in Section 3. Afterwards, we discuss the results of our approach on the MNIST, CIFAR10 and ImageNet datasets in Section 4. We evaluate the influence of the previously described techniques on existing approaches and with our proposed model based on dense shortcut connections. The results show that we can reach state-of-the-art results for existing architectures and improve results even further with our proposed model. Finally, we examine future ideas and conclude our work in Section 5.

Summarized, our contributions in this paper are:

- We present a simple training strategy for binary models without using a pretrained full-precision model.
- We provide empirical evidence that this strategy does not benefit from other commonly used methods, *e.g.*, scaling factors or usage of custom gradient calculation.
- We show that increasing the number of shortcut connections improves the classification accuracy of BNNs significantly and show a novel way to create efficient binary models based on dense shortcut connections.
- We reach state-of-the-art accuracy compared to other approaches for different model architectures and sizes.

2. Related Work

In this section we present related work for binarization and compression techniques.

There are two main approaches which allow for execution on mobile devices by accelerating inference: On the one hand, information in a CNN can be compressed through compact network design. These designs use full-precision floating point numbers as weights, but reduce the total number of parameters and operations through clever network design, while preventing loss of accuracy. On the other hand, information can be compressed by avoiding the common usage of full-precision floating point weights, which use 32 bits of storage. Instead, quantized floating-point numbers with lower precision (*e.g*. 4 bit of storage) or even binary (1 bit of storage) weights are used in these approaches.

First, we present a selection of techniques which utilize the former method. The first of these approaches, *SqueezeNet*, was presented by Iandola *et al*. [9]. The authors replace a large portion of 3×3 filters with smaller 1×1 filters in convolutional layers and reduce the number of input channels to the remaining 3×3 filters for a reduced number of parameters. Additionally, they facilitate late downsampling to maximize their accuracy and use *deep compression* [3] for an overall model size of 0.5 MB.

A different approach, *MobileNets*, was implemented by Howard *et al*. [6]. They use a depth-wise separable convolution where convolutions apply a single 3×3 filter to each input channel. Subsequently, a 1×1 convolution is applied to combine their outputs. Zhang *et al*. [19] use channel shuffling to achieve group convolutions in addition to depth-wise convolution. Their *ShuffleNet* achieves comparably lower error rate for the same number of operations needed for *MobileNets*. These approaches reduce memory requirements, but still require GPU hardware for efficient training and inference. A strategy to accelerate the computation of all these methods for CPUs has yet to be developed.

In contrast to this, approaches which use binary weights instead of full-precision weights achieve compression and acceleration. However, the drawback usually is a severe drop in accuracy. These approaches are based on *Binarized Neural Networks*, introduced by Hubara *et al*. [8], where weights and activations are restricted to +1 and -1. They provide efficient calculation methods for the equivalent of a matrix multiplication by using xnor and $\mathrm{popcount}$ operations. *XNOR-Nets*, published by Rastegari *et al*. [16], improved the performance of binary neural networks by introducing changes to the network layout. Furthermore, they include a channel-wise scaling factor to reduce the approximation error of full-precision weights. Another approach, called *DoReFa-Net*, was presented by Zhou *et al*. [20]. They focus on quantizing the gradients together with different bitwidths (down to binary values) for weights and activations and replace the channel-wise scaling factor with one constant scalar for all filters. A different attempt to strictly use nothing except binary weights is taken in *ABC-Nets* by Lin *et al*. [14]. They use 3 to 5 binary weight bases to approximate full-precision weights. This approximation increases model complexity and size, but reduces the gap between the accuracy of full-precision and binary networks to 5%. Wan *et al*. [17] improved accuracy by using binary weights and ternary activations in their *Ternary-Binary Network*. They train their model from scratch, but they have more operations compared to fully binary models (without an increase in memory consumption). In *Bi-Real Net*, Liu *et al*. [15] modify the *ResNet* architecture by adding additional shortcuts and reducing the size of the convolution layers. They propose a change of gradient computation during backpropagation compared to other approaches. *Bi-Real Nets* are trained using a complex training strategy to fine-tune a pretrained full-precision network to create a binary model with 56.4% accuracy. Our work differs from their approach, as we directly train a binary network from scratch.

3. Methodology

In this section we first provide the major implementation principles of the framework we use for implementing and training binary models. Following this, we examine the usage of scaling factors. Finally, we discuss design principles for binary network layouts and introduce a novel binary model architecture based on *DenseNets*.

3.1. Implementation of Binary Layers

Our implementation is based on the BMXNet framework first presented by Yang *et al*. [18], which itself is based on the MXNet framework. We use the sign function for activation, thus transforming floating-point values into binary values:

$$\mathrm{sign}(x) = \begin{cases} +1 \text{ if } x \geq 0, \\ -1 \text{ otherwise.} \end{cases} \tag{1}$$

The implementation uses a Straight-Through Estimator (STE) [5] with the addition, that it cancels the gradients, when the inputs get too large, as proposed by Hubara *et al*. [8]. The gradient canceling helps the optimization process, since backpropagation no longer increases the absolute value of an input larger than the clipping threshold (which has no actual effect on the loss because sign does not depend on the absolute value). Let c denote the objective function, r_i be a real number input, and $r_o \in \{-1, +1\}$ a binary output. Furthermore, t_{clip} is the threshold for clipping gradients, which was set to $t_{\text{clip}} = 1$ in previous works [8, 20]. Then, the resulting STE is:

$$\text{Forward: } r_o = \operatorname{sign}(r_i)\ . \tag{2}$$

$$\text{Backward: } \frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o} 1_{|r_i| \leq t_{\text{clip}}}\ . \tag{3}$$

Liu *et al.* [15] claim that a tighter approximation, called $\operatorname{approxsign}$, can be made by replacing the backward pass with

$$\frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o} 1_{|r_i| \leq t_{\text{clip}}} \cdot \begin{cases} 2 - 2r_i \text{ if } r_i \geq 0, \\ 2 + 2r_i \text{ otherwise.} \end{cases} \tag{4}$$

Since this could also benefit when training a binary network from scratch, we evaluated this in our experiments.

A large amount of calculations in full-precision networks is usually spent on calculating dot products of matrices, as needed for fully connected and convolutional layers. The computational cost of binary neural networks can be highly reduced by using the xnor and $\operatorname{popcount}$ CPU instructions, first presented by Rastegari *et al.* [16]. They show that the matrix multiplication of a binary input x and weight w can be replaced as follows (n is the number of weights):

$$x \cdot w = 2 \odot \operatorname{bitcount}(\operatorname{xnor}(x', w')) - n\ . \tag{5}$$

Note, that x' and w' are converted from x and w by replacing $\{-1, +1\}$ with $\{0, 1\}$. This means normal training methods with GPU acceleration (*e.g.* cuDNN implementation) can be used (the left side of Equation 5). Afterwards, we can take advantage of the fast CPU implementation with xnor and $\operatorname{popcount}$ (the right side of Equation 5) without any accuracy loss (an example can be found in the supplementary material). To further speedup the final CPU implementation the adjustment by the number of weights can be learned during training (derived from Equation 5):

$$\frac{x \cdot w + n}{2} = \operatorname{bitcount}(\operatorname{xnor}(x', w'))\ . \tag{6}$$

Further, we decide to use no weight decay during training. This was done in previous work before without much explanation [15], so we add our rationale here: Since gradient canceling already prevents the network from optimizing to absolute values larger than the clipping threshold (i.e. the values are already optimal for the current mini-batch), adding weight decay would move these weights away from their optimal values.

3.2. Scaling Methods

In this section, we discuss the usage of a scaling factor during training. Binarization will always introduce an approximation error compared to a full-precision signal. In their analysis, Zhou *et al.* [21] show that this error linearly degrades the accuracy of a CNN. One way to reduce the approximation error, is to use scaling factors [20, 16, 15]. Generally, they follow the structure as in Figure 1.

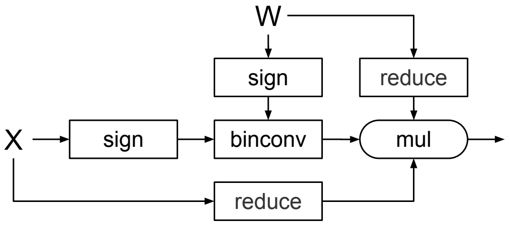


Figure 1: General computation graph for a scaled binary convolution. *binconv* is a convolution based on the optimized operation in Equation 5. *reduce* computes a scaling factor for activations or weights (can be chosen differently), and *mul* is a multiplication operation.

Rastegari *et al.* [16] choose $\operatorname{reduce}(w) = f_{s_w}(w) = \frac{1}{n}||w||_{1,1}$ for each weight filter w. They further propose an efficient method for scaling each feature (i.e. $\operatorname{reduce}(x) = \mathbf{K}$ referring to their paper [16]).

In contrast, Zhou *et al.* [20] reported that a filter-wise weight scaling does not yield improvements. They use one scalar for all weight filters instead, allowing them to also use a binary convolution in the backward pass. Liu *et al.* [15] suggest to use the weight scaling f_{s_w} only in the backward pass to achieve *magnitude aware gradients*.

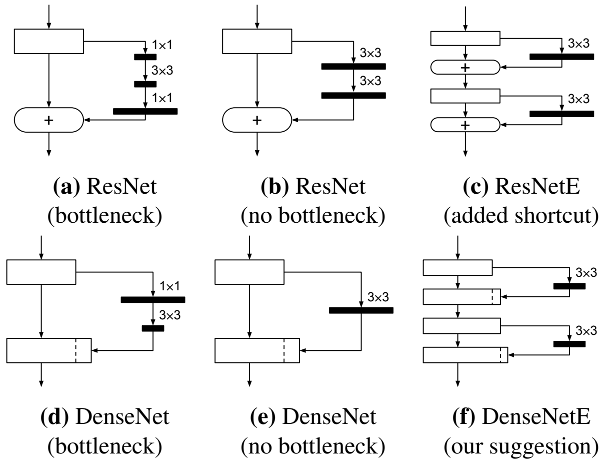
The scaling factors should help binary convolutions to increase the value range. Producing results closer to those of full-precision convolutions and reducing the approximation error. However, these different scaling values influence specific output channels of the convolution. Therefore, a BatchNorm [10] layer directly after the convolution (which is used in *ResNet* and *DenseNet* architectures) theoretically minimizes the difference between a binary convolution with scaling and one without.

Thus, we hypothesize that learning a useful scaling factor is made inherently difficult by BatchNorm layers. We empirically evaluated this in our experiments (see Section 4.1), but want to note that this reasoning might not apply if a binary model is fine-tuned from a full-precision model.

3.3. Network Architectures

In this section we describe general concepts for binary deep neural network architectures first. Afterwards, we show details about *ResNet* [4] and our suggested binary *DenseNet* architecture [7].

Before thinking about model architectures, we must consider the main drawbacks of binary neural networks. First of all, the information density is theoretically 32 times lower, compared to full-precision networks. Research suggests, that the difference between 32 bits and 8 bits seems to be minimal and 8-bit networks can achieve almost identical



(a) ResNet (bottleneck) **(b)** ResNet (no bottleneck) **(c)** ResNetE (added shortcut)

(d) DenseNet (bottleneck) **(e)** DenseNet (no bottleneck) **(f)** DenseNetE (our suggestion)

Figure 2: A single building block of different network architectures (the length of bold black lines represents the number of filters). (a) The original *ResNet* design features a bottleneck architecture. A low number of filters reduces information capacity for binary neural networks. (b) A variation of the *ResNet* architecture without the bottleneck design. The number of filters is increased, but with only two convolutions instead of three. (c) The *ResNet* architecture with an additional shortcut, first introduced by Liu *et al.* [15]. (d) The original *DenseNet* design with a bottleneck in the second convolution operation. (e) The *DenseNet* design without a bottleneck. The two convolution operations are replaced by one 3×3 convolution. (f) Our suggested change to a *DenseNet* where a convolution with N filters is replaced by two layers with $\frac{N}{2}$ filters each.

accuracy as full-precision networks [3]. However, when decreasing bit-width to four or even one bit (binary), the accuracy drops significantly [8, 20]. Therefore, the precision loss needs to be alleviated through other techniques, for example by increasing information flow through the network. We identified three main methods, which help to preserve information despite binarization of the model:

First, a binary model should use as many shortcut connections as possible in the network. These connections allow layers later in the network to access information gained in earlier layers despite of information loss through binarization. Such shortcut connections were proposed for full-precision model architectures in Residual Networks [4] and Densely Connected Networks [7]. Furthermore, this means increasing the number of connections between layers should lead to better model performance, especially for binary networks.

Secondly, following the same idea, network architectures including bottlenecks are always a challenge to adopt. A bottleneck architecture reduces the number of filters and values significantly between the layers, resulting in less information flow through binary neural networks. Therefore we hypothesize, that either we need to eliminate the bottleneck parts or at least increase the number of filters in these bottleneck parts for binary neural networks to achieve best results.

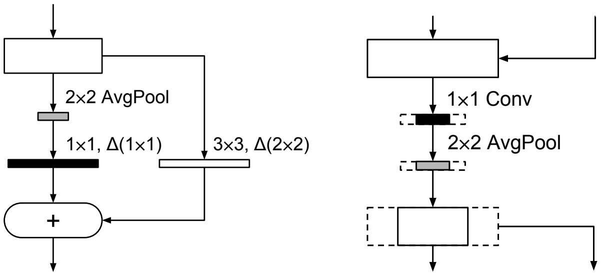
The third way to preserve information (thus increasing model accuracy) comes from replacing certain crucial layers in a binary network with full precision layers. The reasoning is as follows: If layers are binarized, which do not have a shortcut connection, the information lost (due to binarization) can not be recovered in subsequent layers of the network. This affects the first (convolutional) layer and the last layer (a fully connected layer which has a number of output neurons equal to the number of classes). These layers generate the initial information for the network or consume the final information for the prediction, respectively. Therefore, we use full-precision layers for the first and the final layer for all network architectures. We follow authors of previous work on this decision [16, 20], who have empirically shown that binarizing these layers decreases accuracy by a large margin and that the saving of memory and operations is minimal. Another crucial part of deep networks is the downsampling convolution which converts all previously collected information of the network to smaller feature maps with more channels (this convolution often has stride two and output channels equal to twice the number of input channels). Any information lost in this downsampling process is effectively no longer available. Therefore, it should always be considered whether these downsampling layers should be replaced with full-precision layers, even though it increases model size and number of operations.

In the following sections we show how all three methods are applied to a *ResNet* (seen in previous work) and how we applied them to a *DenseNet*.

3.3.1 ResNet Architecture

The *ResNet* architecture, introduced by He *et al.* [4], was the first model architecture that allowed to train models with 18 or more (up to 152) layers. *ResNet* models combine the information of all previous layers with shortcut connections. This is done by adding the input of a block to its output with an identity connection. Consequently, these shortcut connections add no extra weights and very little computational cost, while leading to more meaningful gradients in deeper layers. The bottleneck of a *ResNet* can be removed by replacing the three convolution layers (kernel sizes 1, 3, 1) of a regular *ResNet* block with two three by three convolution layers with a higher number of filters (see Figure 2a, b).

Increasing the number of connections can be done by reducing the block size from two convolutions per block to



(a) ResNet **(b)** DenseNet

Figure 3: The downsampling layers of *ResNet* and *DenseNet*. The bold black lines mark the downsampling layers which can be replaced with full-precision layers. If we use this full-precision layer in a *DenseNet*, we increase the reduction rate to reduce the number of channels (the dashed lines depict the number of channels without reduction).

one convolution per block, as proposed by Liu *et al.* [15]. This leads to twice the amount of shortcuts, as there are as many shortcuts as blocks, if the amount of layers is kept the same (see Figure 2c). However, their method also incorporates other changes to the *ResNet* architecture. Therefore we call this specific change in the block design *ResNetE* (short for Extra shortcut).

Their second change was replacing the downsampling convolution layer (see Figure 3a). This was first proposed by Rastegari *et al.* [16], but neither work quantifies the exact accuracy gain nor the impact on the model size of this design choice.

3.3.2 DenseNet Architecture

DenseNets, proposed by Huang *et al.* [7], use shortcut connections that, contrary to *ResNets*, concatenate the input of a block to its output (see Figure 2d, b). Therefore, new information gained in one layer can be reused throughout the entire depth of the network. To reduce the total model size, the original full-precision architecture includes a bottleneck design for each block and additionally reduces the number of channels in transition layers. This effectively keeps the network at a significantly smaller total size, even though the concatenation adds new information into the network every layer. The number of newly appended features is called growth rate (k) and Huang *et al.* [7] use $k = 32$. The bottleneck of the *DenseNet* architecture can be modified by replacing the two convolution layers (kernel sizes 1 and 3) with one 3×3 convolution (see Figure 2d, e).

However, our experiments showed that reusing the full-precision *DenseNet* architecture for binary neural networks does not achieve satisfactory performance, even after this change. There are different possibilities to increase the capacity of a binary *DenseNet* architecture. The growth rate can be increased (*e.g.* $k = 64, k = 128$), we can use a larger number of blocks, or a combination of both. Both individual approaches add roughly the same amount of parameters to the network. To keep the number of parameters equal for a given *DenseNet* we can halve the growth rate and double the number of blocks at the same time (see Figure 2f) or vice versa. We assume that in this case increasing the number of blocks should provide better results compared to increasing the growth rate. This assumption is derived from our second hypothesis: favoring an increased number of connections over simply adding weights. Similar to a *ResNet* we refer to this adjustment as the *DenseNetE* architecture. However, we note that the actual number of layers and growth rate can be chosen rather freely and evaluate different configurations.

Finally, another characteristic difference of a *DenseNet* compared to a *ResNet* is that the downsampling layer reduces the number of channels [7]. Our experiments showed, that without adjusting the architecture in these downsampling layers, a binary *DenseNet* achieves results of less than 40% accuracy on ImageNet. To preserve information flow in these parts of the network we found two options: On the one hand, we can use no reduction at all, or at least use a lower reduction rate (using a higher number of channels compared to a full-precision architecture). Since the number of channels is initially low in the first downsampling layer (*e.g.* 384 for $k = 128$), we do not need to reduce the number of channels in the first transition layer. However, in the later parts of the network the filter number is higher (*e.g.* 640 at the second transition for $k = 128$), so we use a slight reduction of 1.4 to keep the model size similar to a binary *ResNetE* of equal complexity. On the other hand, we can replace the binary layer in this downsampling layer with a full-precision one (see Figure 3b). Since the full-precision convolution preserves more information, we can use reduction rates equal to (or even higher than) the reduction rate 2 of a full-precision *DenseNet* for all downsampling layers. These higher reduction rates also reduce the number of full-precision (and binary) weights and operations through the whole network, thus allowing us to reach a similar (or even lower) model size compared to the first approach.

Because of the previous reasons, we coupled the decision whether to use a binary or a full-precision downsampling convolution with the choice of reduction rate. The two variants we compare in our experiments (see Section 4) are thus called *full-precision downsampling with high reduction* (halve the number of channels in all transition layers) and *binary downsampling with low reduction* (no reduction in the first transition, divide number of channels by 1.4 in the second and third transition).

Table 1: Evaluation of our binary model performance on the MNIST and CIFAR-10 data sets compared to the results of Yang *et al*. [18].

	Architecture	Model size	Accu-racy	Acc. ([18])
MNIST	LeNet	202KB	**99.0%**	97%
CIFAR-10	ResNetE-18	1.39MB	**87.6%**	86%
CIFAR-10	DenseNetE-21	1.49MB	**90.3%**	-

4. Experiments and Discussion

Following the structure of the previous section, we provide our experimental results to analyze our method with respect to different parameters and techniques. We apply classification accuracy as the general measurement to evaluate the different architectures, methods etc. For brevity, the term *accuracy* always refers to the Top-1 accuracy, unless otherwise noted. Also, differences in accuracies will be noted as $x\%$, but refer to percent point differences. We use the MNIST [13], CIFAR-10 [12] and ImageNet [1] datasets in terms of different levels of task complexity. The experiments were performed on a work station with an Intel(R) Core(TM) i9-7900X CPU, 64 GB RAM and 4×GeForce GTX1080Ti GPUs. All models are trained with the Adam optimizer [11] with an initial learning rate (alpha) of 10^{-2} for CIFAR-10 and 10^{-3} for ImageNet. We trained our ImageNet models for 40 or 50 epochs, and multiply the learning rate by 0.1 at epochs 34 and 37, or epochs 40 and 45 respectively. We use a Gaussian distribution to initialize the weights in the network according to the method proposed by Glorot and Bengio [2].

First, we show the results of a binary *LeNet* for the MNIST dataset and a binary *ResNetE-18* and a *DenseNetE-21* in Table 1 compared to the approach of Yang *et al.* [18]. These results prove that our approach and implementation work on simple datasets, such as MNIST and CIFAR-10, and can reach favorable results compared to other work with the same approach and the same architecture. Moreover, they reveal promising results with our proposed *DenseNetE* architecture, since the model size is increased by only 0.1MB for a 2.7% increase in accuracy.

In the following sections, we first evaluate and discuss the influence of using scaling factors and the approxsign function in the backward pass of the activations for the *ResNetE* network. Following this, we evaluate the impact of the amount of blocks for our proposed *DenseNet* architecture. Then, the design choice of using binary or full-precision downsampling layers for *DenseNet* and *ResNetE* models is empirically verified. Furthermore, we show how the bit-width of downsampling layers changes model performance.

Table 2: The influence of using scaling, a full-precision downsampling convolution, and the approxsign function on the CIFAR-10 dataset based on a *ResNetE-18*. Using approxsign instead of sign slightly boosts accuracy, but only if training a model with scaling factors.

Use scaling of [16]	Downsampl. convolution	Use approxsign of [15]	Accuracy Top1/Top5
no	binary	yes	84.9%/99.3%
		no	87.2%/**99.5%**
	full-precision	yes	86.1%/99.4%
		no	**87.6%/99.5%**
yes	binary	yes	84.2%/99.2%
		no	83.6%/99.2%
	full-precision	yes	84.4%/99.3%
		no	84.7%/99.2%

Table 3: The influence of using scaling, a full-precision downsampling convolution, and the approxsign function on the ImageNet dataset based on a *ResNetE-18*.

Use scaling of [16]	Downsampl. convolution	Use approxsign of [15]	Accuracy Top1/Top5
no	binary	yes	54.3%/77.6%
		no	54.4%/77.5%
	full-precision	yes	56.6%/**79.3%**
		no	**56.7%**/79.2%
yes	binary	yes	53.3%/76.4%
		no	52.7%/76.1%
	full-precision	yes	55.3%/78.3%
		no	55.6%/78.4%

4.1. Scaling Methods

In this section, we discuss the influence of scaling factors (as proposed by Rastegari *et al.* [16]) on the accuracy of our trained models based on the *ResNetE* architecture. First, the results of our CIFAR-10 experiments verify our hypothesis, that applying scaling when training a model from scratch does not lead to better accuracy (see Table 2). All models show a decrease of accuracy between 0.7% and 3.6% when applying scaling factors. Secondly, we evaluated the influence of scaling for the ImageNet dataset (see Table 3). The result is similar, applying scaling reduces model accuracy ranging from 1.0% to 1.4%. We conclude that the scaling is ineffective and suspect two arguments for this: the model can not learn a useful scaling factor when training from scratch or the BatchNorm layers following each convolution layer absorb the effect of the scaling factors. If the first reason applies this is a limitation of our approach of training from scratch, and might not apply to trainings

Table 4: The accuracy of different binary *DenseNet* models by successively splitting blocks evaluated on ImageNet. As the number of connections increases, the model size (and number of binary operations) changes marginally, but the accuracy increases significantly.

Blocks (layers)	Growth-rate	Model size (binary)	Accuracy Top1/Top5
8 (13)	256	3.31 MB	50.2%/73.7%
16 (21)	128	3.39 MB	52.7%/75.7%
32 (37)	64	3.45 MB	**54.3%/77.3%**

Table 5: The difference of performance for different binary *DenseNet* models when using different downsampling methods (see Section 3.3.2) evaluated on ImageNet.

Blocks (layers), growth-rate	Model size (binary)	Downsampl. convolution, reduction	Accuracy Top1/Top5
16 (21), 128	3.39 MB	binary, low	52.7%/75.7%
	3.03 MB	FP, high	**55.9%/78.5%**
32 (37), 64	3.45 MB	binary, low	54.3%/77.3%
	3.08 MB	FP, high	**57.1%/80.0%**

Table 6: Comparison on the ImageNet dataset [1] of our proposed network with binary downsampling branches (see Section 3.3) to ABC-Net [14], which uses this design choice as well.

Model	Our result (model size)	ABC-Net [14]
ResNet-18	54.4%/77.5% (3.36 MB)	42.7%/67.6%
DenseNet	54.3%/77.3% (3.45 MB)	-
ResNet-34	58.1%/80.6% (4.59 MB)	-

based on fine-tuning a full-precision model. If the latter reason applies it should neither increase nor decrease accuracy, which is what we can see for CIFAR-10 (but not for ImageNet) and might still help approaches which are based on fine-tuning.

4.2. Backward Pass of the Sign Function

In this section, we discuss the influence of the backward pass used for the sign function. We compared the regular backward pass, called sign, with the adapted backward pass, called approxsign (see Section 3.1). First, the results of our CIFAR-10 experiments seem to depend on whether we use scaling or not. If we use scaling, both functions perform similarly (see Table 2). Without scaling the approxsign function leads to less accurate models on CIFAR-10.

In our experiments on ImageNet, the performance difference between the use of the functions is minimal (see Table 3). Using one scaling method over the other gives no significant change in model accuracy with one exception: the usage of the sign function results in an accuracy increase of 0.6% if we use scaling and no full-precision shortcut. Therefore, we conclude that applying the approxsign function instead of the sign function seems to be specific to fine-tuning from full-precision models.

4.3. Splitting Layers of DenseNet

We tested our proposed architecture change by comparing *DenseNet* models with varying growth rates and number of blocks (and thus layers). The results show, that increasing the number of connections by adding more layers over simply increasing growth rate increases accuracy in an efficient way (see Table 4). Doubling the number of blocks and halving the growth rate leads to an accuracy gain ranging from 1.4% to 2.5%. However, it seems to have diminishing returns, and training of very deep binary *DenseNet* becomes slow, since less of the calculations can be parallelized. We note that during inference on low-powered devices this is less of a problem compared to training, since the total number of operations is similar between the models (and no additional memory is needed during inference for storing intermediate results, *e.g.* the outputs of the sign function). Therefore, we have not trained even more highly connected models, but highly suspect that this would increase accuracy even further. The total model size slightly increases, since every second half of a split block has slightly more inputs compared to those of a double-sized normal block. In conclusion, our technique of increasing number of connections is highly effective and size-efficient for a binary *DenseNet*.

4.4. Downsampling Layers

We evaluated the difference between using binary and full-precision downsampling layers for both *ResNet* and *DenseNet*. First, we examine the results of *ResNetE-18* on CIFAR-10. Using full-precision downsampling over binary leads to an accuracy gain between 0.3% and 2.3% (see Table 2). However, the model size also increases by 0.64 MB from 1.39 MB to 2.03 MB, which is is arguably too much for this minor increase of accuracy. Our results on the *ResNet* architecture show a significant difference on ImageNet (see Table 3). The accuracy increases by 2% when using full-precision downsampling. Similar to CIFAR-10, the model size increases by 0.64 MB, in this case from 3.36 MB to 4.0 MB. The larger base model size makes the relative model size difference lower and provides a stronger argument for this trade-off. We conclude that the increase in accuracy is significant, especially for ImageNet. However, in our opinion, it does not seem to be large enough to just neglect to acknowledge the significant increase in model size.

Table 7: Comparison of our methods to state-of-the-art binary models on the ImageNet dataset [1]. All these methods use full-precision weights in the convolution layers of the downsampling branches (see Section 3.3).

Model	Our result (model size)	BiReal-Net [15]	TBN [17]	XNOR-Net [16]	Full-precision
ResNet-18	**56.9%/79.7%** (4.0 MB)	56.4%/79.5%	55.6%/74.2%	51.2%/73.2%	69.3%/89.2%
DenseNet	**58.6%/81.0%** (3.99 MB)	-	-	-	-
ResNet-34	60.0%/82.0% (5.23 MB)	**62.2%/83.9%**	58.2%/81.0%	-	73.3%/91.3%

In the following we present our results of a binary *DenseNet* when using a full-precision downsampling with high reduction over a binary downsampling with low reduction. The results of a binary *DenseNet-21* with growth rate 128 for CIFAR-10 result show an accuracy increase of 2.7% from 87.6% to 90.3%. The model size increases from 673 KB to 1.49 MB. This is an arguably sharp increase in model size, but the model is still smaller than a comparable *ResNet-18* with a much higher accuracy. The results of two *DenseNet* architectures (16 and 32 blocks combined with 128 and 64 growth rate respectively) for ImageNet show an increase of accuracy ranging from 2.8% to 3.2% (see Table 5). Further, because of the higher reduction rate, the model size decreases by 0.36 MB at the same time. This shows a higher effectiveness and efficiency of using a full-precision downsampling layer for a *DenseNet* compared to a *ResNet*.

4.5. Comparison to State-of-the-art Approaches

We evaluated our overall approach of training from scratch for a *ResNetE-18*, a *ResNetE-34* and our new architecture *DenseNetE*. Following our results on the influence of the downsampling convolution, we split the comparison between architectures with a full-precision and a binary downsampling convolution.

First, we would like to present the results for models with a binary downsampling convolution (see Table 6). In this case, we use the best *DenseNetE-37* model with the highest number of connections as shown in our previous experiments (see Section 4.3). It has a size comparable to that of a *ResNet-18*. We recognize that our training strategy of training from scratch leads to excellent results compared to the *ABC-Net* [14] approach for both *ResNets* with 18 and 34 layers. The accuracy of our *DenseNetE-37* is close to that of a *ResNet* (with a difference of 0.2%), but it does not improve accuracy. This shows that the techniques applied to a *ResNetE* and our *DenseNetE-37* already successfully increase accuracy by a large margin, even without using full-precision downsampling layers.

Secondly, we also examine the results of models with full-precision downsampling layers (see Table 7). We chose a growth rate of 160 and a reduction rate of 2.2 for a *DenseNetE-21* to match the model size and complexity of a *ResNetE-18* as closely as possible (3.99 MB and 4 MB respectively). Our results show, that we can achieve results similar to a *BiReal-Net* [15] for 18 layers. The accuracy is even slightly (0.5%) higher, even though *BiReal-Net* is trained with a more complex training strategy. But, when we compare a *ResNet-E* trained from scratch to a *BiReal-Net* with 34 layers, we see that accuracy of our approach is 2% lower in comparison. Inspecting our training loss had us suspect, that this gap could be reduced by adapting our choice of optimizer, learning rates, and training for more epochs, but did not want to change this choice to keep our own results comparable. Moreover, our proposed *DenseNetE-21* model reaches 58.6% (an overall improvement of 2.2% over *BiReal-Net-18*) with the same model size. We conclude that accurate binary models can be successfully trained from scratch and do not necessarily need to use a fine-tuning strategy based on pretrained full-precision models.

5. Conclusion

In this paper, we presented our strategy to train binary neural networks from scratch. We clearly separated the existing techniques to increase the number of connections of a binary *ResNet* model and applied them to derive an accurate binary model based on a *DenseNet* architecture. Moreover, we showed the influence of these different techniques through comprehensive experiments and compared our approach to other state-of-the-art approaches. We concluded that accurate binary models can be successfully trained from scratch and our proposed binary architecture even surpasses the state-of-the-art accuracy. However, larger models can still benefit from complex fine-tuning strategies on pretrained full-precision models.

As future work, we would like to examine whether it is possible to better quantify the degree of importance of a layer in the network regarding the preservation of information. Algorithmic approaches for this problem have already been proposed [22]. However, the theoretical knowledge would provide the basis to develop new architectures which benefit from this kind of model quantization. Such a kind of novel architectures could help to reduce the accuracy gap between binary and full-precision layers.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 6, 7, 8

[2] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 6

[3] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. pages 1–14, 2015. 2, 4

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3, 4

[5] G. Hinton. Neural Networks for Machine Learning, Coursera. *URL: http://coursera.org/course/neuralnets (last accessed 2018-03-13)*, 2012. 2

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. 2

[7] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017. 3, 4, 5

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016. 2, 3, 4

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. pages 1–13, 2016. 2

[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015. 3

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[12] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research), 2014. 6

[13] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. 6

[14] X. Lin, C. Zhao, and W. Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 344–352, 2017. 2, 7, 8

[15] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *The European Conference on Computer Vision (ECCV)*, September 2018. 2, 3, 4, 5, 6, 8

[16] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 2, 3, 4, 5, 6, 8

[17] D. Wan, F. Shen, L. Liu, F. Zhu, J. Qin, L. Shao, and H. Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. In *The European Conference on Computer Vision (ECCV)*, September 2018. 2, 8

[18] H. Yang, M. Fritzsche, C. Bartz, and C. Meinel. Bmxnet: An open-source binary neural network implementation based on mxnet. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 1209–1212. ACM, 2017. 2, 6

[19] X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. pages 1–10, 2017. 2

[20] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. 1(1):1–14, 2016. 2, 3, 4

[21] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard. Adaptive Quantization for Deep Neural Network. 2017. 3

[22] Y. Zhou, S. M. Moosavi Dezfooli, N.-M. Cheung, and P. Frossard. Adaptive quantization for deep neural network. In *AAAI*, number CONF, 2018. 8