BinaryDenseNet: Developing an Architecture for Binary Neural Networks

Joseph Bethge, Haojin Yang, Marvin Bornstein, Christoph Meinel
Hasso Plattner Institute, University of Potsdam, Germany
{joseph.bethge,haojin.yang,meinel}@hpi.de, {marvin.bornstein}@student.hpi.de

Abstract

Binary Neural Networks (BNNs) show promising progress in reducing computational and memory costs, but suffer from substantial accuracy degradation compared to their real-valued counterparts on large-scale datasets, e.g., ImageNet. In this work we study existing BNN architectures and revisit the commonly used technique to include scaling factors. We suggest several architectural design principles for BNNs, based on our studies on architectures. Guided by our principles we develop a novel BNN architecture BinaryDenseNet, which is the first architecture specifically created for BNNs to the best of our knowledge. In our experiments, BinaryDenseNet achieves 18.6% *and* 7.6% *relative improvement over the well-known XNOR-Network and the current state-of-the-art Bi-Real Net in terms of top-1 accuracy on ImageNet, respectively. Further, we show the competitiveness of our BinaryDenseNet regarding memory requirements and computational complexity.*
https://github.com/hpi-xnor/BMXNet-v2

1. Introduction

Convolutional Neural Networks (CNNs) have achieved state-of-the-art on a variety of tasks related to computer vision, for example, classification [19], detection [7], and text recognition [17]. By reducing memory footprint and accelerating inference, there are two main approaches which allow for the execution of neural networks on devices with low computational power, *e.g.*, mobile or embedded devices: a CNN can be compressed through compact network design [12, 15, 32] or by avoiding the common usage of full-precision floating point weights and activations, which use 32 bits of storage. Instead, quantized floating-point numbers with lower precision (*e.g.* 4 bit of storage) [33] or even binary (1 bit of storage) weights and activations [14, 21, 24, 25] are used in these approaches. By using binary weights and inputs a BNN can achieve up to $32\times$ memory saving and $58\times$ speedup on CPUs by representing both weights and activations with binary values [25]. However, they still suffer from noticeable accuracy degradation.

Previous approaches focused on reducing quantization errors by using approximation methods [25], scaling factors [25], multiple weight/activation bases [21], fine-tuning a full-precision model [24], multi-stage pre-training [24], or custom gradients [24] (a detailed review of related work can be found in Section 2). These works used well-known real-valued network architectures such as *AlexNet*, *GoogLeNet* or *ResNet* for BNNs without thorough explanation or experiments on the design choices. However, they do not answer the simple yet essential question: *Which architectural design principles are essential for efficient and accurate BNNs?*

Even though large breakthroughs of the field of full-precision CNNs came from designing or discovering new architectures, *e.g.*, ResNet [10] or NASNet [36], not much work has explored the direction of designing an architecture optimal for BNNs. To approach the subject, we first revisit the details of BNNs and study how previous work utilized the *ResNet* architecture (see Section 3).

Following these studies, we suggest architectural design principles and demonstrate them by designing a new BNN architecture *BinaryDenseNet* (see Section 4). In our comparison to previous BNNs, we show that our *BinaryDenseNet* achieves state-of-the-art regarding the general evaluation measures for new architectures (which have been used in previous work [10, 24, 36]): accuracy, model size, and number of operations (see Section 5). Further, we conclude our work in Section 6 with a future outlook.

Summarized, our main contributions in this paper are:

- We empirically and theoretically study previous architecture design choices and the usage of scaling factors.
- We suggest several general design principles for BNNs and further propose a new BNN architecture *BinaryDenseNet*, which significantly surpasses all existing 1-bit CNNs for image classification.
- To guarantee the reproducibility, we contribute to an open source framework for BNN/quantized NN. We share code and models implemented in this paper for classification and object detection. Additionally, we implemented the most influential BNNs including [14, 21, 24, 25, 33] to facilitate follow-up studies.

Table 1: A general comparison of the most related methods to this work. Essential characteristics such as value space of inputs and weights, numbers of multiply-accumulate operations (MACs), numbers of binary operations, theoretical speedup rate and operation types, are depicted. The results are based on a **single quantized convolution layer** from each work. β and α denote the full-precision scaling factor used in proper methods, whilst m, n, k denote the dimension of weight ($W \in \mathbb{R}^{n\times k}$) and input ($I \in \mathbb{R}^{k\times m}$). The table is adapted from [30].

Methods	Inputs	Weights	MACs	Binary Operations	Speedup	Operations
Full-precision	$\mathbb{R}$	$\mathbb{R}$	n×m×k	0	1×	mul,add
BC [3]	$\mathbb{R}$	$\{-1, 1\}$	n×m×k	0	$\sim$ 2×	sign,add
BWN [25]	$\mathbb{R}$	$\{-\alpha, \alpha\}$	n×m×k	0	$\sim$ 2×	sign,add
TTQ [35]	$\mathbb{R}$	$\{-\alpha^n, 0, \alpha^p\}$	n×m×k	0	$\sim$ 2×	sign,add
DoReFa [33]	$\{0, 1\}$×4	$\{0, \alpha\}$	n×k	8×n×m×k	$\sim$ 15×	and,bitcount
HORQ [20]	$\{-\beta, \beta\}$×2	$\{-\alpha, \alpha\}$	4×n×m	4×n×m×k	$\sim$ 29×	xor,bitcount
TBN [30]	$\{-1, 0, 1\}$	$\{-\alpha, \alpha\}$	n×m	3×n×m×k	$\sim$ 40×	and,xor,bitcount
XNOR [25]	$\{-\beta, \beta\}$	$\{-\alpha, \alpha\}$	2×n×m	2×n×m×k	$\sim$ 58×	xor,bitcount
BNN [14]	$\{-1, 1\}$	$\{-1, 1\}$	0	2×n×m×k	$\sim$ 64×	xor,bitcount
Bi-Real [24]	$\{-1, 1\}$	$\{-1, 1\}$	0	2×n×m×k	$\sim$ 64×	xor,bitcount
Ours	$\{-1, 1\}$	$\{-1, 1\}$	0	2×n×m×k	$\sim$ 64×	xor,bitcount

2. Related work

In this section, we roughly divide the recent efforts for binarization and compression into three categories: (i) compact network design, (ii) networks with quantized weights, (iii) and networks with quantized (or binary) weights and activations.

Compact Network Design. This sort of methods uses full-precision floating point numbers as weights, but reduce the total number of parameters and operations through compact network design, while minimizing loss of accuracy. The commonly used techniques include replacing a large portion of 3×3 filters with smaller 1×1 filters [15]; using depth-wise separable convolution to reduce operations [12]; and utilizing channel shuffling to achieve group convolutions in addition to depth-wise convolution [32]. These approaches 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.

Quantized Weights and Real-valued Activations. Recent efforts from this category, for instance, include *BinaryConnect* (BC) [3], *Binary Weight Network* (BWN) [25], and *Trained Ternary Quantization* (TTQ) [35]. In these works, network weights are quantized to lower precision or even binary. Thus, considerable memory saving with minimal accuracy loss has been achieved. But, no noteworthy acceleration can be obtained due to the real-valued inputs.

Quantized Weights and Activations. On the contrary, approaches adopting quantized weights and activations can achieve both compression and acceleration. Remarkable attempts include *DoReFa-Net* [33], *High-Order Residual Quantization* (HORQ) [20] and SYQ [6], which reported promising results on ImageNet [4] with 1-bit weights and multi-bits activations.

Binary Weights and Activations. BNN is the extreme case of quantization, where both weights and activations are binary. Hubara *et al*. proposed *Binarized Neural Network* (BNN) [14], where weights and activations are restricted to +1 and -1. They provide efficient calculation methods for the equivalent of matrix multiplication by using xnor and bitcount operations. *XNOR-Net* [25] improved the performance of BNNs by introducing a channel-wise scaling factor to reduce the approximation error of full-precision parameters. *ABC-Nets* [21] used multiple weight bases and activation bases to approximate their full-precision counterparts. Despite the promising accuracy improvement, the significant growth of weight and activation copies offsets the memory saving and speedup of BNNs. Wang *et al*. [30] attempted to use binary weights and ternary activations in their *Ternary-Binary Network* (TBN). They achieved a certain degree of accuracy improvement with more operations compared to fully binary models. In *Bi-Real Net*, Liu *et al*. [24] proposed several modifications on *ResNet*. They achieved state-of-the-art accuracy by applying an extremely sophisticated training strategy that consists of full-precision pre-training, multi-step initialization (ReLU→leaky clip→clip [23]), and custom gradients.

Table 1 gives a thorough overview of the recent efforts in this research domain. We can see that our work follows the most straightforward binarization strategy as BNN [14], that achieves the highest theoretical speedup rate and the highest compression ratio. However, in contrast to previous work, we directly train a binary network from scratch with a standard training strategy and focus on the understanding of previous architectures and design principles for new binary architectures.

3. Study on Previous Approaches

In this section, to ease the understanding, we first provide a brief overview of the major implementation principles of a binary layer. Afterwards, we study the previously used *ResNet* with regards to binary neural networks. We show that the architectural decision to leave certain layers in full-precision is essential for accurate BNNs on complex tasks. The findings from this study motivates us to explore other effective architectural solutions for creating accurate BNNs. Furthermore, we revisit scaling factors [20, 21, 25, 29, 30, 33, 35] a commonly used techniques in BNNs. Since we did not observe accuracy gain as expected, we analyze why scaling might not be as effective as previously presented and provide empirical proof.

3.1. Implementation of Binary Layers

We apply the sign function for binary activation, thus transforming floating-point values into binary values:

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

The implementation uses a Straight-Through Estimator (STE) [1] with the addition, that it cancels the gradients, when the inputs get too large, as proposed by Hubara *et al*. [14]. 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 work [14, 33]. Then, the resulting STE is:

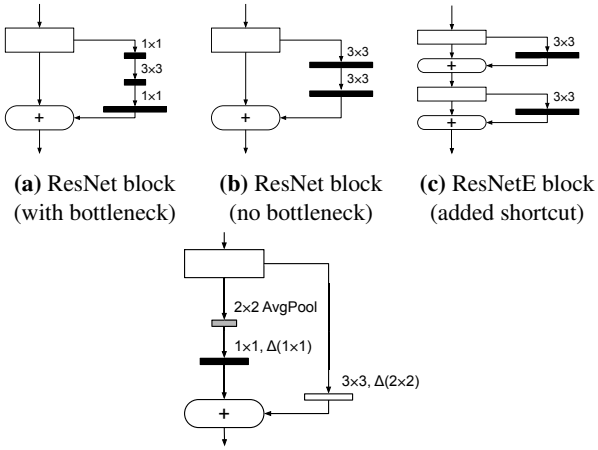
$$\text{Forward: } r_o = \operatorname{sign}(r_i) \ . \tag{2}$$

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

Gradient canceling assists in the optimization process, since backpropagation no longer increases the absolute value of an input larger than the clipping threshold, which is similar to regularization effects in full-precision networks. Furthermore, the computational cost of binary neural networks can be highly reduced by using the xnor and $\mathrm{popcount}$ CPU instructions, as presented by Rastegari *et al*. [25]. 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 divided by the number of input channels):

$$x \cdot w = 2 \odot \operatorname{bitcount}(\operatorname{xnor}(x', w')) - n \ . \tag{4}$$

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 4). Afterwards, we can take advantage of the fast CPU implementation with xnor and $\mathrm{popcount}$ (the right side of Equation 4) without any accuracy loss.



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

(d) Downsampling convolution in a ResNet

Figure 1: Building blocks and a downsampling convolution of a *ResNet* (the length of bold black lines represents the number of filters). (a) The original *ResNet* design features a bottleneck, where a low number of filters reduces information capacity for BNNs. (b) A variation of the *ResNet* without the bottleneck design. The number of filters is increased, but with only two convolutions instead of three. (c) The *ResNetE* architecture with an additional shortcut, first introduced in [24]. (d) The downsampling part with a 1×1 downsampling convolution (black line) which can be either full-precision or binary.

3.2. ResNetE: a ResNet with Extra Shortcuts

A *ResNet* [10] combines 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. Previous work [24, 25, 31] has used *ResNet* architectures with a low amount of layers (18 or 34) for BNNs. Even though the reason seems clear (to keep model size and number of operations low), the implications were not discussed in previous work. By not using larger *ResNet* architectures, these binary models also did not use a bottleneck architecture (see Figure 1a compared to b). In fact, to the best of our knowledge no architecture, which uses bottleneck blocks, has been successfully used for BNNs before.

Furthermore, Liu *et al*. [24] increased the number of connections by reducing the block size from two convolutions per block to one convolution per block. 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 1c). However, Liu *et al*. also incorporate other changes to the *ResNet* architecture. Therefore we call this specific change in the block design *ResNetE* (short for **E**xtra shortcut). Their second change is using a full-precision (instead of binary) downsampling convolution layer, which we discuss in the following section.

Table 2: The influence of using scaling and a full-precision downsampling convolution on the CIFAR-10 and ImageNet datasets based on a binary *ResNetE18*. All models were trained with the same hyperparameters and for either 150 epochs (CIFAR-10) or 40 epochs (ImageNet).

Downsampl. convolution	Scaling of [25]	Top1/Top5 Acc. CIFAR-10	Top1/Top5 Acc. ImageNet
binary	yes	83.6%/99.2%	52.7%/76.1%
	no	87.2%/99.4%	54.5%/77.8%
(model sizes)		*1.39 MB*	*3.36 MB*
full-precision	yes	84.7%/99.2%	55.6%/78.4%
	no	**87.6%/99.5%**	**56.7%/79.2%**
(model sizes)		*2.03 MB*	*4.0 MB*

3.3. Full-precision Layers in Binary Networks

Another method to achieve reasonable accuracy with binary neural networks is replacing certain crucial layers in a binary network with full precision layers. The reasoning is as follows: If layers that do not have a shortcut connection are binarized, 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), as learned from previous work [25, 33, 24, 30, 14]. These layers generate the initial information for the network or consume the final information for the prediction, respectively. Therefore, full-precision layers for the first and the final layer are always applied previously. 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) (see Figure 1d). Any information lost in this downsampling process is effectively no longer available. Some previous work ([24, 25]) has kept these layers in full-precision in addition to the first and last layer. However, the importance and existence of this architectural change does not seem to be well known. It only became apparent to us, when studying the published code of Liu *et al*. and is only shown in an extended technical report [23]. Therefore, we also conducted an ablation study for testing the exact accuracy gain and the impact on the model size.

First, we examine the results of binary *ResNetE18* on CIFAR-10. Using full-precision downsampling over binary leads to an accuracy gain between 0.4% and 1.1% (see Table 2). However, the model size also increases from 1.39 MB to 2.03 MB, which is arguably too much for this minor increase of accuracy. Our results show a more significant difference on ImageNet: the accuracy gain is between 2.2% and 2.6% when using full-precision downsampling. Similar to CIFAR-10, the model size for ImageNet increases by 0.64 MB, in this case from 3.36 MB to 4.0 MB (and 1.06% of the total operations become full-precision). The larger base model size and the low number of affected operations make the relative model size difference lower and provides a stronger argument for this trade-off. We conclude that the increase in accuracy is still significant, especially for ImageNet.

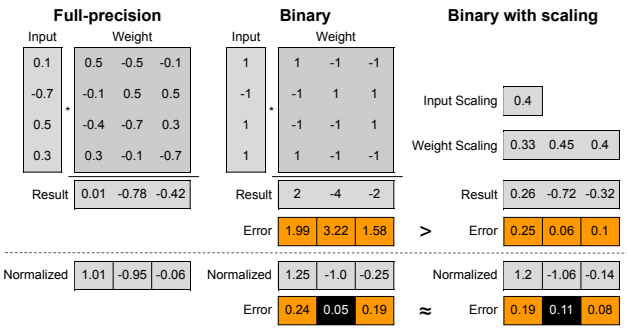


Figure 2: An exemplary implementation shows that normalization minimizes the difference between a binary convolution with scaling (right column) and one without (middle column). In the top row, the columns from left to right respectively demonstrate the *gemm* results of full-precision, binary, and binary with scaling. The bottom row shows their results after normalization. Errors are the absolute difference between full-precision and binary results. The results indicate that normalization dilutes the effect of scaling.

3.4. Scaling Methods

Binarization will always introduce an approximation error compared to a full-precision signal. In their analysis, Zhou *et al*. [34] show that this error linearly degrades the accuracy of a CNN.

Consequently, Rastegari *et al*. [25] propose to scale the output of the binary convolution by the average absolute weight value per channel (α) and average absolut activation over all input channels (K).

$$\mathbf{x} * \mathbf{w} \approx \text{binconv}(\text{sign}(x), \text{sign}(w)) \cdot K \cdot \alpha \qquad (5)$$

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 [16] layer directly after the convolution (which is used in all modern 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. Figure 2 demonstrates an exemplary implementation of our hypothesis.

Table 3: Comparison of our binary *ResNetE18* model (trained for up to 120 epochs) to state-of-the-art binary models using ResNet18 on the ImageNet dataset. The top-1 and top-5 validation accuracy are reported. For the sake of fairness we use the ABC-Net result with 1 weight base and 1 activation base in this table.

Downsampl. convolution	Size	Our result	Bi-Real [24]	TBN* [30]	XNOR [25]	ABC-Net (1/1) [21]
full-precision	4.0 MB	**58.1%/80.6%**	56.4%/79.5%	55.6%/74.2%*	51.2%/73.2%	n/a
binary	3.4 MB	**54.5%/77.8%**	n/a		n/a	42.7%/67.6%

* Even though TBN does not use full-precision downsampling, it is not directly comparable to binary downsampling either, since they use three values for activations.

We empirically evaluated the influence of scaling factors (as proposed by Rastegari *et al*. [25]) on the accuracy of our trained models based on the binary *ResNetE* architecture (see Section 3.2). First, the results of our CIFAR-10 [19] 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 in accuracy between 0.7% and 3.6% when applying scaling factors. Secondly, we evaluated the influence of scaling for the ImageNet dataset. The result is similar, scaling reduces model accuracy ranging from 1.0% to 1.7%. We conclude that either the BatchNorm layers following each convolution layer absorb the effect of the scaling factors or the benefit of using scaling factors only exists, when using a pre-trained full-precision model instead of training from scratch. To avoid the additional computational and memory costs, we do not use scaling factors in the rest of the paper.

4. Developing a New Architecture

In this section, we present several design principles for developing accurate architectures for BNNs. These are derived from our studies on the binary *ResNetE* model in the previous section. Afterwards, we propose a new BNN model *BinaryDenseNet* based on these insights, which has a higher number of shortcut connections and reaches state-of-the-art accuracy.

4.1. Golden Rules for Architecture Design

As shown in Table 3, with a standard training strategy our binary *ResNetE18* model outperforms other state-of-the-art binary models simply by using the same architectural changes of previous work. Inspired by this success, we suggest several general design principles for BNN architectures, summarized as follows:

- The core of our theory is *maintaining rich information flow of the network*, which can effectively compensate the precision loss caused by quantization.
- Not all the well-known real-valued network architectures can be seamlessly applied for BNNs. The network architectures from the category *compact network design* are not well suited for BNNs, since their design philosophies are mutually exclusive (eliminating redundancy $\leftrightarrow$ compensating information loss).
- *Bottleneck* design [28] should be eliminated in your BNNs. We will discuss this in detail in the following paragraphs (also confirmed by [2]).
- Consider using full-precision downsampling layer in your BNNs to preserve the information flow for complex tasks.
- Using shortcut connections is a straightforward way to avoid bottlenecks of information flow, which is particularly essential for BNNs.
- To overcome bottlenecks of information flow, we should appropriately increase the network width (the dimension of feature maps) while going deeper (as *e.g*., see *BinaryDenseNet37/37-dilated/45* in Table 6). However, this may introduce additional computational costs.

Before thinking about new model architectures, we must consider the main drawbacks of BNNs. 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 accuracy as full-precision networks [9]. However, when decreasing bit-width to four or even one bit (binary), the accuracy drops significantly [14, 33]. Therefore, the precision loss needs to be alleviated through other techniques, for example by increasing information flow through the network. We further describe the three main methods we learned from our studies in the previous chapter, 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 precision loss through binarization. Furthermore, this means that increasing the number of connections between layers should lead to better model performance, especially for binary networks.

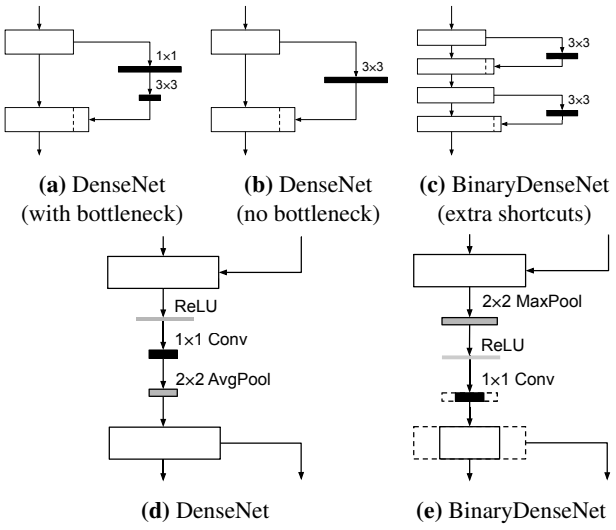


Figure 3: Different building blocks and downsampling convolutions of the *DenseNet* and *BinaryDenseNet* architecture (the length of bold black lines represents the number of filters). (a) The original *DenseNet* design with a bottleneck. (b) The *DenseNet* design without a bottleneck. The two convolution operations are replaced by one 3×3 convolution. (c) Our suggested change to a *DenseNet* where a convolution with N filters is replaced by two layers with $\frac{N}{2}$ filters each. (d) The original DenseNet transition block. (e) We changed the layer ordering to effectively reduce the number of MACs. If we use full-precision downsampling in a *BinaryDenseNet*, we increase the reduction rate to reduce the number of channels (the dashed lines depict the number of channels without reduction).

Secondly, following our ideas in Section 3.2, network architectures including bottlenecks are always a challenge to adopt. The bottleneck design reduces the number of filters and values significantly between the layers, resulting in less information flow through BNNs. Therefore we hypothesize that we either need to eliminate the bottlenecks or at least increase the number of filters in these bottleneck parts for BNNs to achieve best results.

Thirdly, as discussed in Section 3.3 certain crucial layers in a binary network should be replaced with full precision layers to preserve information. The first and the final layer should always be left in full-precision. Additionally, for complex tasks, the *downsampling convolution* layers in a network should be kept in full-precision.

4.2. BinaryDenseNet

DenseNets [13] apply shortcut connections that, contrary to *ResNet*, concatenate the input of a block to its output (see Figure 3a, b). Therefore, new information gained in one layer can be reused throughout the entire depth of the network. We believe this is a significant characteristic for maintaining information flow. Thus, we construct a novel BNN architecture: *BinaryDenseNet*.

The bottleneck design and transition layers of the original *DenseNet* effectively keep the network at a smaller total size, even though the concatenation adds new information into the network every layer. However, as previously mentioned, we have to eliminate bottlenecks for BNNs. The bottleneck design can be modified by replacing the two convolution layers (kernel sizes 1 and 3) with one 3×3 convolution (see Figure 3a, b). However, our experiments showed that *DenseNet* architecture does not achieve satisfactory performance, even after this change. This is due to the limited representation capacity of binary layers. There are different ways to increase the capacity. We can increase the growth rate parameter k, which is the number of newly concatenated features from each layer. We can also use a larger number of blocks. Both individual approaches add roughly the same amount of parameters to the network. To keep the number of parameters equal for a given *BinaryDenseNet* we can halve the growth rate and double the number of blocks at the same time (see Figure 3c) 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 hypothesis: favoring an increased number of connections over simply adding weights.

Another characteristic difference of *BinaryDenseNet* compared to binary *ResNetE* is that the downsampling layer reduces the number of channels. To preserve information flow in these parts of the network we found two options: On the one hand, we can use a full-precision downsampling layer, similarly to binary *ResNetE*. Since the full-precision layer preserves more information, we can use higher reduction rate for downsampling layers. To reduce the number of MACs, we modify the transition block by swapping the position of pooling and convolution layers. We use *MaxPool* $\rightarrow$ *ReLU* $\rightarrow$ 1×1-*Conv* instead of *ReLU* $\rightarrow$ 1×1-*Conv* $\rightarrow$ *AvgPool* in the transition block (see Figure 3e, d). On the other hand, we can use a binary downsampling convlayer instead of a full-precision layer with a lower reduction rate, or even no reduction at all. 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 4.3) 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 4: The difference of performance for different *BinaryDenseNet* models when using different downsampling methods evaluated on ImageNet.

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

Table 5: The accuracy of different *BinaryDenseNet* 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	Growth-rate	Model size (binary)	Accuracy Top1/Top5
8	256	3.31 MB	50.2%/73.7%
16	128	3.39 MB	52.7%/75.7%
32	64	3.45 MB	**55.5%/78.1%**

4.3. Ablation Studies

Downsampling Layers. In the following we present our evaluation results of a *BinaryDenseNet* when using a full-precision downsampling with high reduction over a binary downsampling with low reduction. The results of a *BinaryDenseNet21* with growth rate 128 for CIFAR-10 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 binary *ResNet18* with a much higher accuracy. The results of two *BinaryDenseNet* 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 4). 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 FP downsampling layer for a *BinaryDenseNet* compared to a binary *ResNet*. Note, that 2.2% of all operations are in these downsampling layers and they become full-precision operations (for a *BinaryDenseNet28*). Therefore, we compared the effiency of our architectures to previous work in Section 5.2.

Splitting Layers. We tested our proposed architecture change (see Figure 3c) by comparing *BinaryDenseNet* 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 5). Doubling the number of blocks and halving the growth rate leads to an accuracy gain ranging from 2.5% to 2.8%. Since the training of very deep *BinaryDenseNet* models becomes slow (because of memory requirements, which are not a problem during inference, since the intermediate results are only needed for backpropagation), 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 the number of connections is highly effective and size-efficient for a *BinaryDenseNet*.

Table 6: Comparison of our *BinaryDenseNet* to state-of-the-art 1-bit CNN models on ImageNet.

Model size	Method	Top-1/Top-5 accuracy
∼4.0MB	XNOR-ResNet18 [25]	51.2%/73.2%
	TBN-ResNet18 [30]	55.6%/74.2%
	Bi-Real-ResNet18 [24]	56.4%/79.5%
	BinaryResNetE18	58.1%/80.6%
	BinaryDenseNet28	**60.7%/82.4%**
∼5.1MB	TBN-ResNet34 [30]	58.2%/81.0%
	Bi-Real-ResNet34 [24]	62.2%/83.9%
	BinaryDenseNet37	62.5%/83.9%
	BinaryDenseNet37-dilated[*]	**63.7%/84.7%**
7.4MB	*BinaryDenseNet45*	63.7%/84.8%
46.8MB	Full-precision ResNet18	69.3%/89.2%
249MB	Full-precision AlexNet	56.6%/80.2%

[*] *BinaryDenseNet37-dilated* is slightly different to other models as it applies dilated convolution kernels, while the spatial dimention of the feature maps are unchanged in the 2nd, 3rd and 4th stage that enables a broader information flow.

5. Comparison to State-of-the-Art

In this section, we report our main experimental results on image classification and object detection using *BinaryDenseNet*. We further report the computation cost in comparison with other quantization methods. Our implementation is based on the *BMXNet* framework first presented by Yang *et al*. [31]. All models are trained with the Adam optimizer [18] with an initial learning rate (alpha) of 10^{-3} for ImageNet. We trained our ImageNet models for up to 120 epochs, and multiply the learning rate by 0.1 at epochs 100 and 110 and use a Gaussian distribution to initialize the weights in the network according to the method proposed by Glorot and Bengio [8]. We use a clipping threshold t_{clip} (see Section 3.1) of 1.25 or 1.3 instead of 1, since experiments showed it increases accuracy by 0.2% to 0.3%.

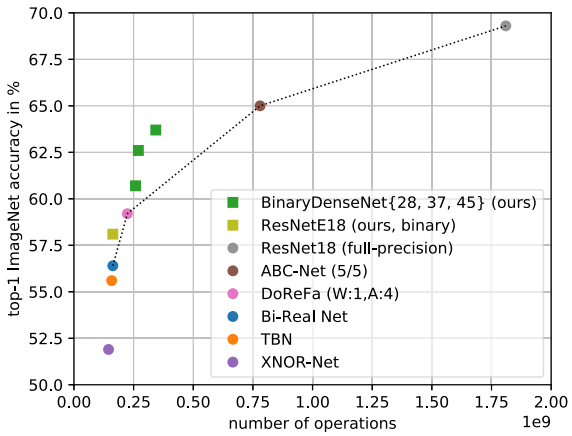


Figure 4: The trade-off of top-1 validation accuracy on ImageNet and number of operations. All the binary/quantized models are based on *ResNet18* except *BinaryDenseNet*.

5.1. Image Classification

To evaluate the classification accuracy, we report our results on ImageNet [4]. Table 6 shows the comparison result of our *BinaryDenseNet* to state-of-the-art BNNs with different sizes. For this comparison, we chose growth and reduction rates between layers for *BinaryDenseNet* models to match the model size and complexity of the corresponding binary *ResNet* architectures as closely as possible (see Table 7 for details). Our results show that *BinaryDenseNet* surpass all the existing 1-bit CNNs with noticeable margin. Particularly, *BinaryDenseNet28* with 60.7% top-1 accuracy, is better than our binary *ResNetE18*, and achieves up to 18.6% and 7.6% relative improvement over the well-known XNOR-Network and the current state-of-the-art Bi-Real Net, even though they use a more complex training strategy and additional techniques, *e.g.*, custom gradients and a scaling variant.

5.2. Efficiency Analysis

For this analysis, we adopted the same calculation method as [10, 24]. They separately sum up the numbers of full-precision (f) and binary operations (b). Then they calculate the number of operations as $f + \frac{1}{32} \cdot b$, based on the theoretical speedup factor of 32. Figure 4 shows that our binary *ResNetE18* demonstrates higher accuracy with the same computational complexity compared to other BNNs. Furthermore, our *BinaryDenseNet28/37/45* achieve significant accuracy improvement with only small additional computation overhead. For a more challenging comparison, we include models with other bit-widths for weights (w) and activations (a): DoReFa-Net (w: 1, a: 4) [33], TBN (w: 1, a: 1.5) [30], and ABC-Net (w: 5, a: 5) [21]. Overall, our *BinaryDenseNet* models show superior performance while measuring both accuracy and computational efficiency.

Table 7: The detailed configuration of all our models. All models use 64 channels in the first convolution and all BinaryDenseNet (BDN) models use a growth-rate of 64 for each block. We show the number of layers for each stage and the reduction rate between each of the four stages.

Model	Layers	Reduction rate	Best/total epochs	t_{clip}
ResNetE18	4, 4, 4, 4	-	115/120	1.25
BDN-28	6, 6, 6, 5	2.7, 2.7, 2.2	117/120	1.3
BDN-37	6, 8, 12, 6	3.3, 3.3, 4	118/120	1.3
BDN-45	6, 12, 14, 8	2.7, 3.3, 4	123/130	1.3

Table 8: Object detection performance in Mean Average Precision (mAP) by using our *BinaryDenseNet37/45* with an SSD 512 [22] on the VOC2007 [5] test set compared to Full-Precision (FP) approaches.

	BinaryDenseNet 37/45 (SSD 512)	SSD 512 FP ([22])	Faster RCNN FP ([27])	Yolo FP ([26])
mAP	**66.4/68.2**	76.8	73.2	66.4

5.3. Preliminary Result on Object Detection

We adopted the off-the-shelf toolbox Gluon-CV [11] for the object detection experiment. We change the base model of the adopted SSD architecture [22] to *BinaryDenseNet* and train our models on the combination of PASCAL VOC2007 trainval and VOC2012 trainval, and test on VOC2007 test set [5]. Table 8 illustrates the results of our binary SSD as well as full-precision detection models [22, 26, 27].

6. Conclusion

We discovered, that previous work in the field of BNNs mostly concentrated on training improvements unrelated to the architecture. Therefore, there was little understanding on the effects of architectural changes and adaptations of the *ResNet* architecture for BNNs. In our work, we provided empirical and theoretical studies of previous architecture changes and the common usage of scaling factors. We derived architectural design principles and demonstrated that they can be used to create an efficient novel *BinaryDenseNet* architecture. We showed, that our new architecture achieves state-of-the-art accuracy with regards to model size and number of operations compared to previous work on ImageNet. Although the task is still arduous, we hope the ideas and results of this paper will provide new potential directions for the future development of BNNs. In future work, we would like to explore other architecture changes, which could improve accuracy and efficiency of binary models even further.

References

[1] Y. Bengio, N. Léonard, and A. C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

[2] J. Bethge, H. Yang, C. Bartz, and C. Meinel. Learning to train a binary neural network. *CoRR*, abs/1809.10463, 2018.

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

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[5] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[6] J. Faraone, N. Fraser, M. Blott, and P. H. Leong. Syq: Learning symmetric quantization for efficient deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

[8] 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.

[9] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[10] 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.

[11] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li. Bag of tricks for image classification with convolutional neural networks. *arXiv preprint arXiv:1812.01187*, 2018.

[12] 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. *arXiv preprint arXiv:1704.04861*, 2017.

[13] 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.

[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in neural information processing systems*, 2016.

[15] 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 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[16] 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.

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman. Deep features for text spotting. In *Computer Vision – ECCV 2014*, pages 512–528, Cham, 2014. Springer International Publishing.

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[19] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10. *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010.

[20] Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao. Performance guaranteed network acceleration via high-order residual quantization. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

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

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. In *ECCV*, pages 21–37, 2016.

[23] Z. Liu, W. Luo, B. Wu, X. Yang, W. Liu, and K. Cheng. Bi-real net: Binarizing deep network towards real-network performance. *CoRR*, abs/1811.01335, 2018.

[24] 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 *ECCV*, September 2018.

[25] 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.

[26] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788, 2016.

[27] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28*, pages 91–99, 2015.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, and Others. Going deeper with convolutions. Cvpr, 2015.

[29] W. Tang, G. Hua, and L. Wang. How to Train a Compact Binary Neural Network with High Accuracy. *AAAI*, 2017.

[30] 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 *ECCV*, September 2018.

[31] 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.

[32] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[33] 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. *arXiv preprint arXiv:1606.06160*, 2016.

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

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

[36] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.