Truncating Wide Networks using Binary Tree Architectures

Yan Zhang[†], Mete Ozay[†], Shuohao Li[†‡] and Takayuki Okatani[†§]

[†]Tohoku University, [‡]National University of Defense Technology, [§]RIKEN Center for AIP

{zhang,mozay,lishuohao,okatani}@vision.is.tohoku.ac.jp

Abstract

In this paper, we propose a binary tree architecture to truncate architecture of wide networks by reducing the width of the networks. More precisely, in the proposed architecture, the width is incrementally reduced from lower layers to higher layers in order to increase the expressive capacity of networks with a less increase on parameter size. Also, in order to ease the gradient vanishing problem, features obtained at different layers are concatenated to form the output of our architecture. By employing the proposed architecture on a baseline wide network, we can construct and train a new network with same depth but considerably less number of parameters. In our experimental analyses, we observe that the proposed architecture enables us to obtain better parameter size and accuracy trade-off compared to baseline networks using various benchmark image classification datasets. The results show that our model can decrease the classification error of a baseline from 20.43% *to* 19.22% *on Cifar-100 using only* 28% *of parameters that the baseline has. Code is available at* https://github.com/ZhangVision/bitnet.

1. Introduction

Recently, Deep Neural Networks (DNNs) have achieved impressive results for many image classification tasks [5, 6, 14, 18, 23, 25, 26, 27]. Various architectures of DNNs have been proposed in order to improve classification accuracy. One approach used to improve accuracy of a DNN is to widen each layer while keeping the depth unchanged. For instance, it is empirically shown in [28] that a wide but relatively shallow DNN can obtain an accuracy comparable to a narrow but relatively deep DNN on several classification tasks. There are two crucial benefits of wide-shallow DNNs. First, they usually run *faster* than narrow-deep DNNs on parallel computing devices, e.g. GPUs, as illustrated in [28]. Also, a deep DNN with many layers may suffer from the *gradient vanishing problem*. Reducing the depth can ease this problem as shown in [7]. However, parameter size of DNNs may significantly increase with respect to improvement of accuracy by widening each layer.

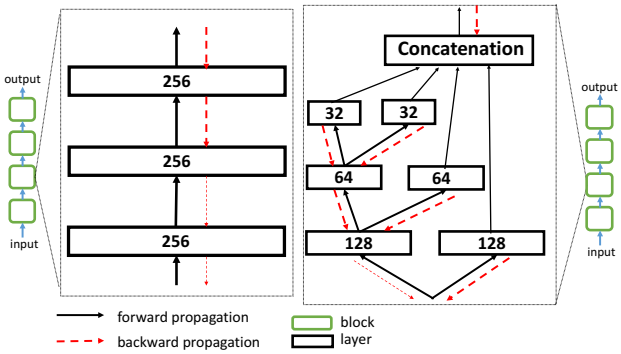

Figure 1. **Left**: A conventional architecture [5]. **Right**: Proposed binary tree architecture. Arrows indicate flow of data. The numbers depicted in the rectangle denote the width of the layer. Depth of both architectures is 3, and their output width is 256. Our architecture has considerably less number of parameters.

In this paper, we address a problem of improvement of the parameter size and accuracy trade-off of the aforementioned wide-shallow DNNs. Specifically, given a baseline wide-shallow DNN, we aim to construct and train a new DNN equipped with the following two desirable properties: 1) The depth of the new DNN is not greater than that of the baseline wide-shallow DNN so that it can keep the aforementioned two benefits, and 2) the new DNN can achieve comparable accuracy using relatively less number of parameters compared to the baseline DNN, or can achieve better accuracy by using the same number of parameters that the baseline DNN has.

Toward this end, we propose a binary tree architecture for implementation of DNNs with a *better* trade-off. An illustrative comparison of our proposed binary tree architecture and conventional architectures [5, 28] is given in Figure 1. In conventional architectures, layers having same width (number of channels) are sequentially stacked. In the proposed binary tree architecture, the width of the k^{th} layer is $\frac{D}{2^{k-1}}$, where D is the width of the first layer ($k = 1$, in-

put layer). Additionally, connections between layers of the proposed architecture are established as connections used in an asymmetric binary tree. At each k^{th} layer of a binary tree architecture, we have $C_k = \frac{D}{2^{k-1}}$ channels. Then, $\frac{C_k}{2}$ of channels are connected to the channels of the $(k+1)^{st}$ layer. In addition, the remaining $\frac{C_k}{2}$ channels are directly concatenated to form the output of the architecture. Note that our binary tree architecture can be generalized to fully connected layers in which the width is equal to the number of neurons used at the layer.

Our **motivation** for employment of the proposed binary tree architecture is twofold. First, we aim to increase the **expressive capacity of DNNs** with a relatively small increase of parameter size. In this paper, we use the definition of expressive capacity proposed in [21, 19], where it is defined to be the maximal number of linear regions of (decision) functions computable by the given DNN. As shown in [21, 19], it reflects the complexity of class decision boundary computable by the DNN. Their results state that the maximal number of linear regions of a fully connected feed forward neural network endowed with ReLU [20] activation functions grows exponentially with respect to the depth of the network, and polynomially with respect to the width of the network (i.e., the number of neurons used at each layer of the network). Following this theoretical result, one can increase the expressive capacity with a small increase of the parameter size by simply stacking more layers with *small* width. This leads to the first characteristic structure of our binary tree architecture, which is obtained by continuous decrease of the width from the input layer to the higher layers by a factor of 2^{-1}. With this specific structure, the expressive capacity grows with small increase of the parameter size. In our experiments, a binary tree used at convolutional layers of a DNN can increase classification accuracy with a small increase of parameter size of the DNN.

The second motivation for employment of our binary tree architecture is to **ease a vanishing gradient problem observed in DNNs**. While training DNNs, the magnitude of gradient can be cumulatively reduced when it is propagated from higher layers to lower layers. Therefore, the more layers are used for propagation, the weaker gradient will be obtained at the lower layers, which makes it difficult to train a DNN with many layers [1, 4]. Consequently, the gradient vanishing problem suggests reduction of the depth of a DNN, while increasing the depth may lead to efficient improvement of the expressive capacity as mentioned above. Thus, we need to trade-off between easing the gradient vanishing problem and increasing the expressive capacity, which motivates our second characteristic structure of our proposed binary tree architecture that is obtained by concatenation of features obtained at different layers. With this specific structure, gradients can propagate through short path to lower layers. An illustration is given in Figure 1, where flow of gradient propagation during backpropagation is depicted by red dash line. For a better illustration of vanishing gradients, we use thicker red line to show stronger gradient. Our empirical analyses given in Section 4.4 show that concatenation of features at lower layers can ease the gradient vanishing problem.

Our contributions are summarized as follows.

(1) We propose a binary tree architecture which can be used to improve the trade-off between parameter size and classification accuracy of baseline wide-shallow DNNs. Meanwhile, the depth of the wide-shallow DNNs is not increased using the proposed architecture in order to keep the benefit of running speed on parallel computing devices.

(2) Our experimental results show that, on the Cifar datasets, one can construct a DNN using the proposed binary tree architecture to achieve better accuracy but using considerably less number of parameters. On the Cifar-100 dataset, our models can outperform corresponding baselines by using only approximately 50% of baseline's parameter size. One of our models decreases the classification error of a baseline from 20.43% to 19.22% by using only 28% of parameters that the baseline has. On ILSVRC12 task, we construct and train two DNNs using the proposed binary tree architecture which also provide better parameter size and classification accuracy trade-off than baseline models.

(3) We also provide a theoretical analysis of the expressive capacity of DNNs endowed with our proposed binary tree architecture as a function of its depth and width. The theoretical results indicate that expressive capacity of DNNs endowed with our proposed binary tree architecture can grow with small increase of the parameter size.

Our paper is organized as follows. The second section provides the related work. In Section 3, we introduce our binary tree architecture. Experimental results and analyses are given in Section 4. Section 5 concludes the paper.

2. Related Work

Recently, various architectures of convolutional neural networks (CNNs) have been proposed [14, 18, 26, 23, 27, 25]. In [8], connections between random forests [2] and CNNs are investigated. Inspired by random forests, they embedded routing functions to CNNs, and obtained Conditional CNNs. As shown in their experiments, Conditional CNNs with highly branched tree architectures can improve the accuracy-efficiency trade-off. Conditional CNNs can be considered as symmetric full tree architectures. On the other hand, we use an asymmetric tree architecture, and concatenate features from different layers. In [15], a *fractal architecture* used by FractalNet was proposed. The relationship between FractalNets and our architecture is as follows. Fractal architecture can also be considered as a tree architecture. In addition, they both aim to ease the gradient vanishing problem by joining lower-layer features to

higher-layer features. The difference is that our architecture aims to reduce the parameter size by incrementally reducing the width as depth increases, while FractalNets do not. As a result, FractalNets have more number of parameters compared to our proposed architecture for the same depth. The motivation for development of our proposed architecture considers the expressive capacity which is not investigated in FractalNets. Experimental results on benchmark datasets also show that proposed architecture outperforms FractalNets by using less number of parameters.

In previous works [10, 12, 16], various tree structures were explored in the context of DNNs. All these related work and our work enable an end-to-end training of tree structures in DNNs. Our method is different from them in the following aspects. In [12], a decision forest classifier was introduced to replace a classifier used at the last classification layer (usually Softmax) of a baseline DNN; other layers, e.g., convolutional layers, were left untouched. Thus, the parameter size was not reduced unlike our method. In [10], encoding and decoding functions of Auto-Encoders were implemented using *soft* decision trees. Their motivation was to recursively direct samples from parent nodes to all its child nodes according to probabilities computed by gating functions, and the output of tree was the average of output of all leaves weighted by gating values. In [16], a tree structure was used in pooling layer, the region being pooled was fed into the leaves of a tree structure, and results were obtained at the root of tree structure. Therefore, they employed a tree architecture different from (in an inverse way to) our tree structure.

As shown in [11, 29], the parameter size of a trained CNN can be reduced by constructing a new CNN with less redundant weights. However, these methods may cause a decrease of accuracy after compression of models. With our architecture, we can boost the accuracy using less number of parameters. In [5], a residual architecture was proposed to construct CNNs (ResNets). ResNets enable us to train considerably deeper CNNs. The deepest ResNets employed for classification using the ILSVRC12 [22] and Cifar datasets had 200 and 1000 layers, respectively [5, 6], and achieved impressive performance. However, a deep ResNet with many layers may suffer from two problems. First, the running speed on parallel computing devices may be slow compared to the speed of shallower ones. Thus, it takes more time to train a very deep ResNet. Also, the gradient vanishing problem [1, 4] may also be observed on a very deep ResNet with many layers. To address these problems, a novel training procedure called stochastic depth was introduced in [7]. It enables one to train shallow ResNets during training, and to use deep ResNets for testing. Shallow ResNets can ease the gradient vanishing problem during training, and reduce the training time. Their experimental results show that their proposed training procedure can improve the test accuracy of baseline ResNets with constant depth. This indicates that easing the gradient vanishing problem is crucial to train a very deep ResNet. However, during the test time, the depth of ResNets is kept as same as the depth of baseline, which decreases inference speed.

Intuitively, a simpler way to avoid gradient vanishing problem and accelerate the running speed during training and inference is to use a shallow network. In [28], Zagoruyko and Komadakis used a shallow ResNet, and increased the width to make the expressive capacity comparable with narrow-deep ones. Interestingly, they observed that wide-shallow CNNs can outperform its deeper-narrower peer CNNs which have the same parameter size on the Cifar-10/100 classification datasets. On the ILSVRC12 classification task, wide ResNets can also obtain comparable accuracy with smaller depth. Their results draw our attention to consider the wide-shallow DNNs. In their method, the width of each layer was symmetrically increased by the same factor. This significantly increases the parameter size. Therefore, in this work, the proposed binary architecture truncate architecture of wide networks by reducing width of the networks considering their parameter size and accuracy trade-off. Our motivation for employment of binary tree architecture also considers the gradient vanishing problem, which has been shown to be crucial for training DNNs in [7].

3. Binary Tree Architecture

In this section, we give an overview of conventional blocks used in previous works [5, 28], and introduce our proposed binary tree architecture. Then, we theoretically analyze expressive capacity and parameter size of our proposed binary tree architecture considering its depth and width.

3.1. Conventional Blocks

In various CNNs [23, 5, 6, 28], a block is constructed by a stack of K convolutional layers. At each k^{th} layer, $k = 1, 2, \ldots, K$, we compute

$$\mathbf{X}_k = f_k(\mathbf{X}_{k-1}; \mathcal{W}_k), \tag{1}$$

where $\mathbf{X}_0 := \mathbf{X} \in \mathbb{R}^{w \times h \times c}$, $\mathbf{X}$ is an input tensor of features given to the block, $\mathbf{X}_k \in \mathbb{R}^{\frac{w}{s} \times \frac{h}{s} \times D}$ is the tensor of features obtained at the output of k^{th} layer, c is the number of channels, w is the $width$ and h is the $height$ of a feature map. We assume that the number of convolutional filters used at each layer is D, and down-sampling is performed at the first convolutional layer by stride s. $f_k(\mathbf{X}_k; \mathcal{W}_k)$ is computed by a composition of a convolution operation, batch normalization [9] and nonlinear function σ, where $\mathcal{W}_k$ denotes a set of trainable parameters used at the k^{th} layer, e.g. the filter weights. The output feature tensor of the block

is obtained at the last layer $\mathbf{Y} := \mathbf{X}_K \in \mathbb{R}^{\frac{w}{s} \times \frac{h}{s} \times D}$. In (1), we omit a shortcut connection and an element-wise addition used in [5] to simplify the notation. We refer to the block defined in (1) as a *conventional block* (ConvenBlock) with width D and depth K in the following sections. An illustration of ConvenBlock with width 256 and depth 3 is given in Figure 1 (left).

3.2. Binary Tree Blocks

We define a *binary tree block* (BitBlock) by a binary tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes residing in the block, and $\mathcal{E}$ is the set of edges that connect the nodes. The architecture of a BitBlock is determined by its width D and depth K, where D is the channel number, i.e. number of feature tensors provided by the BitBlock at the output, and K is the depth of the binary tree $\mathcal{T}$, as follows.

- At the root of a BitBlock $\mathcal{T}$ denoted by $v_0 \in \mathcal{V}$, we have a feature tensor denoted by $\mathbf{X}_{0,l} := \mathbf{X} \in \mathbb{R}^{w \times h \times c}$, where $\mathbf{X}$ is given as an input to the block.

- At each k^{th} layer (level) of a BitBlock, we apply two mapping functions $f_{k,l}$ and $f_{k,r}$ to the input feature tensor $\mathbf{X}_{k-1,l}$. Then, we compute a feature tensor $\mathbf{X}_{k,l}$ having $\frac{D}{2^k}$ channels on a left child node of v_{k-1}, and a feature tensor $\mathbf{X}_{k,r}$ having $\frac{D}{2^k}$ channels on a right child node of v_{k-1} at the k^{th} layer of $\mathcal{T}$. The feature tensor $\mathbf{X}_{k,l}$ is further fed to the next $(k+1)^{st}$ layer. More precisely, at each k^{th} level, we compute

$$\begin{aligned} \mathbf{X}_{k,l} &= f_{k,l}(\mathbf{X}_{k-1,l}; \mathcal{W}_{k,l}), \\ \mathbf{X}_{k,r} &= f_{k,r}(\mathbf{X}_{k-1,l}; \mathcal{W}_{k,r}). \end{aligned} \tag{2}$$

- Finally, feature tensors computed at all right child nodes at each k^{th} level of $\mathcal{T}$, and the feature tensors computed at the left child node at the last K^{th} level of $\mathcal{T}$ are concatenated across channels to construct the output feature tensor of the BitBlock by

$$\mathbf{Y} = \text{concat}(\mathbf{X}_{1,l} \bullet \mathbf{X}_{2,l} \bullet \cdots \bullet \mathbf{X}_{K,l} \bullet \mathbf{X}_{K,r}). \tag{3}$$

  The size of concatenated tensor $\mathbf{Y}$ is $(\frac{w}{s} \times \frac{h}{s} \times D)$. In the case of down-sampling, we apply a stride s at the first layer as utilized in ConvenBlocks.

We note that D is divided by 2^K. Moreover, if $K = 1$, then the BitBlock $\mathcal{T}$ reduces to a single convolution layer. Our BitBlocks can also be applied to fully connected layers using a feature tensor $\mathbf{X} \in \mathbb{R}^{1 \times 1 \times C}$. We denote it by a fully connected BitBlock. An illustration of our proposed BitBlock with width of 256 and depth of 3 is given in Figure 1 (right). We can construct a DNN by stacking multiple BitBlocks. In this paper, a DNN endowed with BitBlocks is called a BitNet.

3.3. A Theoretical Analysis of Expressive Capacity of Binary Tree Blocks

In this section, we theoretically analyze expressive capacity and parameter size of the proposed binary tree block with respect to its depth and width, when it is used at fully connected layers. We use the definition of expressive capacity proposed in the previous work [21, 19]. More precisely, expressive capacity of a DNN is defined by the maximal number of linear regions of (decision) functions computable by the given DNN. The formal definition of linear regions of a function is given as follows.

Definition 3.1 (Linear Region). Given a function $f(\cdot)$ with n-dimensional input space $\mathbb{R}^n$, a linear region $\mathbb{R}^n_i$ of $f(\cdot)$ is a subspace of its input space, such that $f(\cdot)$ computes a linear mapping on that linear region, i.e. $f(\mathbf{x}) := \mathbf{w}_i\mathbf{x} + \mathbf{b}_i$, $\forall \mathbf{x} \in \mathbb{R}^n_i$, and $f(\mathbf{x}) :\neq \mathbf{w}_i\mathbf{x} + \mathbf{b}_i, \forall \mathbf{x} \notin \mathbb{R}^n_i$.

Suppose that a decision function $f(\cdot)$ is computed by a DNN. Then, the more number of linear regions $f(\cdot)$ has, the more complex input-output mapping the DNN can compute. In other words, the DNN can be used to compute more complex decision boundary, and to solve more complex classification tasks. A multilayer perceptron network implementing a linear activation function computes a linear mapping between input and output. Thus, it only has 1 linear region, i.e. the whole input space.

We provide the following two theoretical results regarding i) computation of the parameter size, and ii) computation of the maximal number of linear regions of functions computable by a fully connected conventional network and a BitNet. Proofs of the theorems are given in the supplemental material. We first provide the results for conventional networks that can be easily derived using the results given in [19].

Corollary 3.2. *Suppose that we are given a fully connected neural network stacked by L fully connected ConvenBlocks each having width D and depth K. The parameter size of the given network is $\mathcal{O}\big(LKD^2\big)$. The maximal number of linear regions of functions that can be computed by the given network in an n-dimensional ($D \geq n$) input space is lower bounded by $\mathcal{O}\big((\frac{D}{n})^{nKL}\big)$.* ■

The expressive capacity of our proposed BitNet is given as follows.

Proposition 3.3. *Suppose that we are given a BitNet stacked by L fully connected BitBlocks each having width D and depth K. The parameter size of the given BitNet is $\mathcal{O}\Big(\frac{4}{3}L\big(1-\frac{1}{4^K}\big)D^2\Big)$. The maximal number of linear regions of functions that can be computed by the given BitNet in an n-dimensional ($D \geq 2^K n$) input space is lower bounded by $\mathcal{O}\Big(\big(\frac{D}{2^K n}\big)^{nKL}\Big)$.* ■

As we observe from these theoretical results, if D and L are fixed, then the expressive capacity of BitNets can grow with a small increase of the parameter size as K increases. Although the expressive capacity of a fully connected conventional network can grow faster as K increases, its parameter size also grows faster than that of a BitNet. Although these theoretical results are obtained for fully connected layers, our experimental observations reflect that the results can be applied also for the convolutional layers. In our experiments, we observe that a binary tree used at convolutional layers of a DNN can increase classification accuracy with a small increase of parameter size of the DNN.

4. Experimental Results

We empirically analyze the proposed binary tree architecture using various baseline ResNets and several benchmark image classification datasets. We also analyze the gradient vanishing problem of the proposed binary tree architecture. In the implementation of BitNets, given a baseline ResNet, we can construct a BitNet of the same depth and the same block width with considerably fewer parameters compared to the baseline (ResNet). We can also increase the block width but keep the depth of baseline model using the proposed binary tree architecture, and obtain a BitNet with a similar number of parameters. The configuration details of BitNets are given for each classification task.

4.1. Cifar-10 and Cifar-100

Cifar-10 and Cifar-100 [13] are image datasets consisting of 50,000 training images and 10,000 test images. The spatial size of each image is 32×32. Cifar-10 and Cifar-100 consist of 10 and 100 categories, respectively. The architectures of BitNets and ResNets used to perform analyses on the Cifar datasets are given in Table 1. As we can see from the table, the depth of a net is determined by the number of blocks in each group (n), and the depth of each block (k). The width of each block is denoted by d.

We first compare performance of our proposed BitNets and that of Wide ResNets [28] by constructing nets with same depth and block width. Specifically, given a Wide ResNet with fixed value of d, k, and n, we use the same block width d for a BitNet. Then, we set values of k and n for the BitNet, such that the total depth of the BitNet equals to the total depth of the given Wide ResNet. We can use different value combinations of k and n for the BitNet as long as $k \times n$ value for the BitNet equals to $k \times n$ for the given Wide ResNet. For example, given a 38-layer Wide ResNet with $(d = 4, k = 2, n = 6)$, we can construct a BitNet with $(d = 4, k = 3, n = 4)$ or a BitNet with $(d = 4, k = 4, n = 3)$, both having 38 layers.

For fare comparison with Wide ResNets, we use the same training and testing setting as employed in [28]. The details of the setting are given in the supplemental material.

Group Name	Configuration	Output Size
conv1	conv, 16 channels	32×32
conv2	block$(d \times 16, k) \times n$	32×32
conv3($\downarrow$)	block$(d \times 32, k) \times n$	16×16
conv4($\downarrow$)	block$(d \times 64, k) \times n$	8×8
gap	global average pooling	1×1
fc	10 or 100-way softmax	

Table 1. The CNN architecture employed for classification using the Cifar-10 and Cifar-100. BitNets use BitBlock as block type, and ResNets use ConvenBlock with residual connection. k denotes the depth of each block ($k = 2$ for all ResNets used in this paper). d denotes the width of each block. n denotes a stack of n blocks. All convolutional layers use filters of size 3×3. Batch Normalization is used at every convolutional layer before ReLU. Down-sampling is performed by applying stride 2 at the first convolutional layer of the first block in Group conv3 and conv4.

Table 2 provides the comparative results for several Wide ResNets and BitNets, which are designed using the same width and depth. As we can see from the results, using the same depth and width, BitNets have considerably less number of parameters and FLOPs. Moreover, BitNets can outperform Wide ResNets using considerably less number of parameters. We compare BitNets with four baseline Wide ResNets. In the analyses, we obtained the following results:

(1) A Wide ResNet having $(d = 4, k = 2, n = 6)$ is the deepest and narrowest architecture among four baseline architectures. The BitNet $(d = 4, k = 4, n = 3)$ and the BitNet $(d = 4, k = 3, n = 4)$ can obtain performance comparable to this baseline, and their parameter size is only 30% and 41% of that of this baseline, respectively. The BitNet $(d = 4, k = 2, n = 6)$ is constructed by using larger number of blocks, but reducing the depth of each block. As shown in Section 4.4, this BitNet suffers from the gradient vanishing problem during the training due to use of small k and large n. As a result, the performance slightly degrades. BitNet $(d = 4, k = 6, n = 2)$ has the least parameter size, i.e. has only 19% of the baseline's parameter size. However, it also performs worst.

(2) Compared with the first baseline ResNet $(d = 4, k = 2, n = 6)$, the baseline ResNet $(d = 10, k = 2, n = 2)$ is wider and shallower. The BitNet $(d = 10, k = 2, n = 2)$ outperforms it by 1% for the Cifar-100 dataset by using approximately 56% of the baseline ResNet's parameter size. Another configuration of BitNet $(d = 10, k = 4, n = 1)$ uses only one BitBlock at each group, resulting in only 23% of the baseline's parameter size. However, the performance is also degraded by more than 1% using the Cifar-100 dataset. For the Cifar-10 dataset, both BitNets obtain similar performance compared to the baseline.

(3) For the Wide ResNet $(d = 10, k = 2, n = 3)$, both

Model	Depth	Number of Parameters	FLOPs	Cifar-10	Cifar-100
NIN [18]	-	-	-	8.81	35.67
ELU [3]	-	-	-	6.55	24.28
DSN [17]	-	-	-	7.97	34.57
AllCNN [24]	-	-	-	7.25	33.71
ResNet [5]	1202	10.2M	-	4.91	–
preact-ResNet [6]	1001	10.2M	-	4.62	22.71
Stochastic Depth ResNet [7]	110	1.7M	-	5.25	24.98
FractalNet [15]	40	22.9M	-	5.24	22.49
Wide ResNet (d=4,k=2,n=6) [28]	38	8.9M	1.34×10^9	4.97	22.89
BitNet (d=4,k=3,n=4)	38	3.7M	0.53×10^9	4.82	<u>22.19</u>
BitNet (d=4,k=4,n=3)	38	2.7M	0.39×10^9	<u>4.65</u>	22.60
BitNet (d=4,k=2,n=6)	38	5.4M	0.78×10^9	5.31	23.22
BitNet (d=4,k=6,n=2)	38	1.7M	0.24×10^9	4.77	23.87
Wide ResNet (d=10,k=2,n=2) [28]	14	17.1M	2.64×10^9	4.56	21.59
BitNet (d=10,k=2,n=2)	14	9.6M	1.32×10^9	<u>4.17</u>	<u>20.48</u>
BitNet (d=10,k=4,n=1)	14	3.9M	0.49×10^9	4.97	23.88
Wide ResNet (d=10,k=2,n=3) [28]	20	26.8M	4.06×10^9	4.44	20.75
BitNet (d=10,k=2,n=3)	20	15.6M	2.21×10^9	**<u>3.78</u>**	<u>19.29</u>
BitNet (d=10,k=3,n=2)	20	10.2M	1.41×10^9	3.81	19.37
Wide ResNet (d=12,k=2,n=4) [28]	26	52.5M	7.87×10^9	4.33	20.43
BitNet (d=12,k=2,n=4)	26	31.2M	4.45×10^9	<u>4.07</u>	**<u>19.06</u>**
BitNet (d=12,k=4,n=2)	26	14.9M	2.06×10^9	4.11	19.22

Table 2. Classification error (%) of CNNs for the Cifar-10/100 datasets. Using the same depth and block width, our BitNets can outperform Wide ResNets using considerably less number of parameter size. Underlined numbers indicate the best performance among models having the same depth and same block width. Bold numbers denote the best performance obtained for all models. Definitions of d, k and n are given in Table 1. All of Wide ResNets and our BitNets are trained using data augmentation, and without using dropout.

BitNets obtain more than 1% performance boost using the Cifar-100 dataset. For the Cifar-10 dataset, the performance boost is more than 0.5%. Notably, the parameter size of the BitNet $(d = 10, k = 3, n = 2)$ is only 38% of the parameter size of the baseline.

(4) The Wide ResNet $(d = 12, k = 2, n = 4)$ has the largest parameter size among four baseline models. Our two BitNets both outperform the baseline by more than 1% accuracy. Note that the parameter size of the BitNet $(d = 12, k = 4, n = 2)$ is only 28% of that of the baseline.

We emphasize that the performance of baseline Wide ResNets are already close to the state-of-the-art. Thus, more than 1% boost of the accuracy is obtained. To summarize, most of our BitNets can achieve better or approximately equal accuracy using less number of parameters, which indicates that our binary tree architecture can improve the parameter size and accuracy trade-off of baseline Wide ResNets. There are two BitNets whose accuracy are roughly 1% lower than that of their baselines. This is possibly because they use too less number of BitBlocks causing insufficient expressive capacity. The rest of BitNets obtain sufficient expressive capacity using relatively less number of parameter size. Compared with other previous models, such as Stochastic Depth ResNet [7] and FractalNet [15], our BitNets can outperform them using less number of parameters.

4.2. ILSVRC12

In order to evaluate the proposed architecture on a large scale image classification dataset, we also use the training and validation dataset of ILSVRC12 [22], which consists of 1.3 million training images and 50,000 validation images belonging to 1000 categories. The details of training and testing settings are given in the supplemental material.

In this task, we construct two BitNets (BitNet-26 and BitNet-34) in order to compare their performance with that of ResNet-34 [5]. The details of models are given in Table 3. The classification results are given in Table 4. The results show that our BitNets have better parameter size and accuracy trade-off compared to ResNets. In the results, BitNet-34 outperforms ResNet-34 B by 1% using the same parameter size, while their depth is the same. An-

Group	BitNet-26	BitNet-34	Output Size
conv1	conv 7×7, 64 channels, stride 2		112×112
conv2	max pooling 3×3, stride 2		56×56
	$\mathrm{b}(128, 3) \times 2$	$\mathrm{b}(256, 4) \times 2$	
conv3	$\mathrm{b}(192, 3) \times 1$ $\mathrm{b}(256, 3) \times 1$	$\mathrm{b}(384, 4) \times 2$	28×28
conv4	$\mathrm{b}(384, 3) \times 2$	$\mathrm{b}(512, 4) \times 2$	14×14
conv5	$\mathrm{b}(512, 3) \times 1$ $\mathrm{b}(768, 3) \times 1$	$\mathrm{b}(768, 4) \times 2$	7×7
gap	global average pooling		1×1
fc	1000-way softmax		
Param.	12.79M	22.99M	
FLOPs	2.8×10^9	7.8×10^9	
Depth	26	34	

Table 3. Structures of BitNets used for classification of the ILSVRC12 dataset. $\mathrm{b}(d, k) \times n$ refers to a stack of n BitBlocks with width d and depth k. All convolutional layers employed in each BitBlock use filters of size 3×3. The output size is reduced by applying stride 2 at the first convolutional layer of the first block in some groups.

Model	Single Crop	Ten Crop	Number of Parameters
ResNet-18 B [5]	30.43	28.22	13.1M
Width $\times 1.5$ [28]	27.06	–	25.9M
Width $\times 2$ [28]	25.58	–	45.6M
FractalNet-34 [15]	–	24.12	–
ResNet-34 B [5]	26.73	24.76	23.2M
Width $\times 1.5$ [28]	24.5	–	48.6M
BitNet-26	27.74	25.83	12.8M
BitNet-34	25.46	23.77	23.0M

Table 4. Single model, Top-1 classification error (%) obtained using the ILSVRC12 validation dataset.

other smaller BitNet-26 obtains 1% higher error compared to ResNet-34 B. However it is shallower and its parameter size is approximately 50% of ResNet-34 B. Compared to the FractalNet-34 model, BitNet-34 also outperforms it. BitNet-26 outperforms ResNet-18 B by approximately 2% accuracy using the same number of parameters. Increasing the wide of ResNet-18 by methods given in [28] can improve the accuracy. However, the cost for obtaining the improvement is also huge. BitNet-26 and ResNet-18 B width $\times 1.5$ have comparable accuracy, but the parameter size of our BitNet is almost two times smaller than that of ResNet. Similarly, the parameter size of BitNet-34 is only 50% of ResNet-18 width $\times 2$.

	BitNet $(d = 4, k, n = 4)$		BitNet $(d = 12, k, n = 2)$	
	Param.	Cifar10/100	Param.	Cifar10/100
$k = 1$	2.7M	4.98/23.54	10.3M	6.02/23.80
$k = 2$	3.5M	**4.68**/22.72	13.8M	4.02/19.86
$k = 3$	3.7M	4.82/**22.19**	14.7M	**3.98**/18.97
$k = 4$	3.7M	4.69/22.65	14.9M	4.11/19.22
$k = 5$	3.7M	4.69/22.86	15.0M	4.09/**18.95**
$k = 6$	3.7M	4.77/22.69	15.0M	4.19/19.51

Table 5. Change of classification error (%) of two BitNets with respect to k using the Cifar-10/100 datasets. Definitions of d, n, and k are given in Table 1.

4.3. Experimental Analyses of Effect of Depth of BitBlock to Classification Performance

We observe that using the same width d and the same number of BitBlocks n, BitBlocks having different depth may provide different performance (see BitNet $(d = 10, k = 3, n = 2)$ and BitNet $(d = 10, k = 2, n = 2)$ in Table 2). Thus, we further analyze how performance of BitNets changes with respect to the depth of BitBlocks. For this purpose, we evaluate BitNet $(d = 4, k, n = 4)$ (a deep-narrow one) and BitNet $(d = 12, k, n = 2)$ (a relatively shallower-wider one) by setting different values to k. The results are given in Table 5. As illustrated in the table, as k increases, both BitNets gain a performance boost due to an increase of the expressive capacity. Note that for the BitNet $(d = 12, k, n = 2)$ employed using the Cifar-100, there is almost a 4% performance boost from $k = 1$ to $k = 2$ with a 33% increase of parameter size. We observe that the performance boosts further as k increases. These observations match our theoretical analyses provided in Section 3.3, which states that as d and n are fixed, increasing k can increase the expressive capacity of a BitNet with a small increase of parameter size. However, the boosting trend tends to be saturated if $k \geq 3$, and the increase of parameter size is also negligible. This can be explained as follows. As k increases, the width of convolutional layer also decreases in the binary tree architecture resulting in a saturated expressive capacity. Additionally, we also observe that wider BitNets $(d = 12)$ gain more performance boost than a narrower Bitnet $(d = 4)$ with the same increase on k. This is simply because the increased layers of the wider BitNet are wider than that of the narrower one, thus it can increase the expressive capacity more.

4.4. Experimental Analyses for the Gradient Vanishing Problem

In this section, we analyze our BitNets considering the gradient vanishing problem. During the training of a BitNet or Wide ResNet using the Cifar-100, we compute the mean

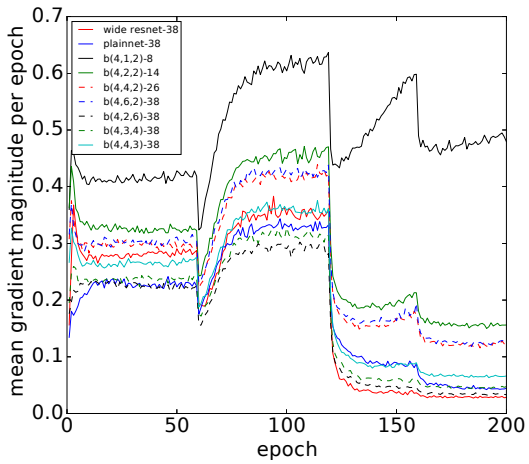

Figure 2. Mean magnitude (L2-norm) of gradients computed per epoch during the training at the first convolutional layer. $\mathrm{b}(d,k,n)-m$ refers to an m-layer BitNet with configuration (d,k,n) defined in Table 1. Best viewed in color print.

magnitude (L2-norm) of gradients obtained at the first convolutional layer per epoch. The results are given in Figure 2. We analyze the results using nine models having different depth but same width $d=4$. In the figure, wide resnet-38 refers to the Wide ResNet $(d=4,k=2,n=6)$ having 38 layers, and plainnet-38 is the one designed without using residual connections. Models denoted by $\mathrm{b}(d,k,n)-m$ are our proposed BitNets where m is the total depth.

As we can see from the figure, for all models having 38 layers, our BitNet $\mathrm{b}(4,6,2)-38$ shows the strongest magnitude, even stronger than wide resnet-38. This result indicates that using concatenation in the proposed binary tree architecture can ease the gradient vanishing problem. We also observe that gradient becomes weaker as the number of blocks n is increased and the depth k of each BitBlock is reduced. For instance, for all 38 layers BitNets, the magnitude can be roughly sorted by $\mathrm{b}(4,2,6)-38$, $\mathrm{b}(4,3,4)-38$, $\mathrm{b}(4,4,3)-38$ and $\mathrm{b}(4,6,2)-38$ according to the increasing strength of the magnitude. This observation reflects that as n increases and k decreases, features obtained at less number of lower layers are concatenated to form the output of each BitBlock. In general, gradients propagate towards more layers to reach lower layers.

We also analyze how magnitudes of gradients change with respect to BitBlock's depth k, when d and n are fixed. In the results, $\mathrm{b}(4,1,2)-8$ provides the strongest magnitude among all nine models as expected, since it is the shallowest model. By increasing k, we observe that the magnitudes computed using BitNet $\mathrm{b}(4,2,2)-14$ and $\mathrm{b}(4,4,2)-26$ are decreased, since more layers are used to propagate gradients in BitBlocks.

Model	Number of Parameters	Error
wide resnet-38	8.9M	22.89
plainnet-38	8.9M	29.13
$\mathrm{b}(4,1,2)-8$	1.2M	29.19
$\mathrm{b}(4,2,2)-14$	1.5M	25.45
$\mathrm{b}(4,4,2)-26$	1.7M	22.72
$\mathrm{b}(4,6,2)-38$	1.7M	23.87
$\mathrm{b}(4,2,6)-38$	5.4M	23.22
$\mathrm{b}(4,3,4)-38$	3.7M	22.19
$\mathrm{b}(4,4,3)-38$	2.7M	22.60

Table 6. Classification error (%) of the models given in Figure 2 obtained using the Cifar-100 dataset.

The errors obtained using these nine models are given in Table 6. Although $\mathrm{b}(4,1,2)-8$ and $\mathrm{b}(4,2,2)-14$ show stronger gradient magnitude than wide resnet-38, they provide higher classification error. This is mainly because the depth of these two BitNets is too small resulting in insufficient expressive capacity. BitNet $\mathrm{b}(4,4,3)-38$ obtains comparable classification performance by using larger depth. The gradient magnitude of BitNet $\mathrm{b}(4,4,3)-38$ is also comparable with that of resnet, which benefits from the proposed binary tree architecture. Without using binary tree architecture, the gradient magnitude of plainnet-38 is weaker and its final classification error is higher compared to that of resnet-38.

5. Conclusions

We introduced a binary tree architecture in order to truncate architecture of wide networks considering their parameter size and accuracy trade-off. In the proposed architecture, the width of each layer of a network is incrementally reduced from lower layers to higher layers of the network. Moreover, features obtained at different layers are concatenated to form the output of our architecture. In the theoretical analysis, we explored the expressive capacity of BitNets. In our experiments, the networks which were designed using the proposed architecture, called BitNets, obtained better parameter size and accuracy trade-off on several benchmark datasets compared to baseline networks endowed with conventional architectures. Additionally, we observed that the concatenation structure can ease the gradient vanishing problem. In our future work, we plan to use BitNets for object detection tasks.

Acknowledgement

This work was partly supported by CREST, JST Grant Number JPMJCR14D1 and JSPS KAKENHI Grant Number JP15H05919.

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[2] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units(elus). In *ICLR*, 2016.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[7] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[8] Y. Ioannou, D. P. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown, and A. Criminisi. Decision forests, convolutional networks and the models in-between. arXiv:1603.01250, 2016.

[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[10] O. Irsoy and E. Alpaydin. Autoencoder trees. In *ACML*, 2014.

[11] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[12] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. B. . Deep neural decision forests. In *ICCV*, 2015.

[13] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[15] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017.

[16] C.-Y. Lee, P. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *AISTATS*, 2016.

[17] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015.

[18] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.

[19] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NIPS*, 2014.

[20] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[21] R. Pascanu, G. Montúfar, and Y. Bengio. On the number of inference regions of deep feed forward networks with piecewise linear activations. In *ICLR*, 2014.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, pages 1–42, April 2015.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[24] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR workshop*, 2015.

[25] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv:1602.07261, 2016.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[28] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.

[29] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *CVPR*, June 2015.