Deep Pyramidal Residual Networks

Dongyoon Han*
EE, KAIST
dyhan@kaist.ac.kr

Jiwhan Kim*
EE, KAIST
jhkim89@kaist.ac.kr

Junmo Kim
EE, KAIST
junmo.kim@kaist.ac.kr

Abstract

Deep convolutional neural networks (DCNNs) have shown remarkable performance in image classification tasks in recent years. Generally, deep neural network architectures are stacks consisting of a large number of convolution layers, and they perform downsampling along the spatial dimension via pooling to reduce memory usage. Concurrently, the feature map dimension (i.e., the number of channels) is sharply increased at downsampling locations, which is essential to ensure effective performance because it increases the diversity of high-level attributes. This also applies to residual networks and is very closely related to their performance. In this research, instead of sharply increasing the feature map dimension at units that perform downsampling, we gradually increase the feature map dimension at all units to involve as many locations as possible. This design, which is discussed in depth together with our new insights, has proven to be an effective means of improving generalization ability. Furthermore, we propose a novel residual unit capable of further improving the classification accuracy with our new network architecture. Experiments on benchmark CIFAR-10, CIFAR-100, and ImageNet datasets have shown that our network architecture has superior generalization ability compared to the original residual networks.

Code is available at https://github.com/jhkim89/PyramidNet

1. Introduction

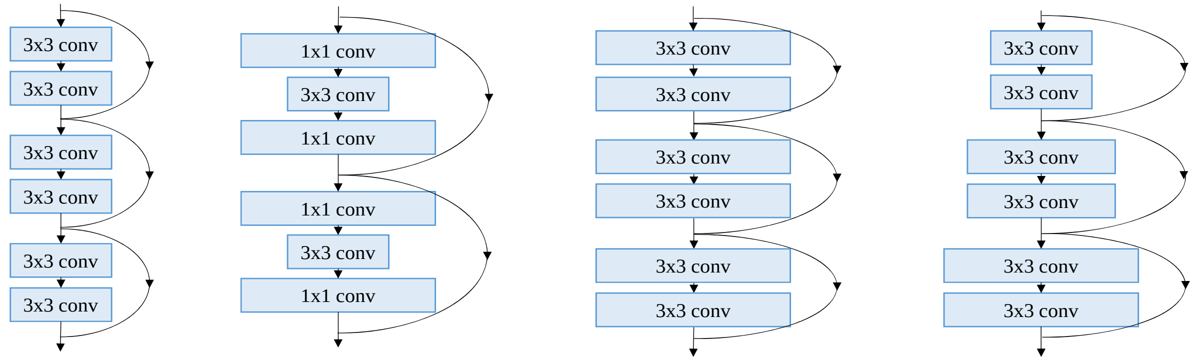
The emergence of deep convolutional neural networks (DCNNs) has greatly contributed to advancements in solving complex tasks [13, 23, 2, 3, 19] in computer vision with significantly improved performance. Since the proposal of LeNet [16], which introduced the use of deep neural network architectures for computer vision tasks, the advanced architecture AlexNet [13] was selected as the winner of the 2012 ImageNet competition [22] by a large margin over traditional methods. Subsequently, ZF-net [35], VGG [25], GoogleNet [31], Residual Networks [7, 8], and Inception Residual Networks [30] were successively proposed to demonstrate advances in network architectures. In particular, Residual Networks (ResNets) [7, 8] leverage the concept of shortcut connections [29] inside a proposed residual unit for residual learning, to make it possible to train much deeper network architectures. Deeper network architectures are known for their superior performance, and these network architectures commonly have deeply stacked convolutional filters with nonlinearity [25, 31].

With respect to feature map dimension, the conventional method of stacking several convolutional filters is to increase the dimension while decreasing the size of feature maps by increasing the strides of the filters or poolings. This is the widely adopted method of controlling the size of feature maps, because extracting the diversified high-level attributes with the increased feature map dimension is very effective for classification tasks. Architectures such as those of AlexNet [13] and VGG [25] utilize this method of increasing the feature map dimension to construct their network architectures. The most successful deep neural network, ResNets [7, 8], which was introduced by He *et al.* [7], also follows this approach for filter stacking.

According to the research of Veit *et al.* [33], ResNets are considered to behave as ensembles of relatively shallow networks. These researchers showed that the deletion of an individual residual unit from ResNets, i.e., such that only a shortcut connection remains, does not significantly affect the overall performance, proving that deleting a residual unit is equivalent to deleting some shallow networks in the ensemble networks. Contrary to this, deleting a single layer in plain network architectures such as a VGG-network [25] damages the network by causing additional severe errors.

However, in the case of ResNets, it was found that deleting the building blocks in a residual unit with downsampling, where the feature map dimension is doubled, still increases the classification error by a significant margin. Interestingly, when the residual net is trained using a stochastic depth [10], it was found that deleting the blocks with downsampling does not degrade the classification performance, as shown in Figure 8 in [33]. One may think that

[*] These two authors contributed equally.



(a) basic (b) bottleneck (c) wide (d) pyramidal

Figure 1. Schematic illustration of (a) basic residual units [7], (b) bottlenecks [7], (c) wide residual units [34], and (d) our pyramidal residual units.

this phenomenon is related to the overall improvement in the classification performance enabled by stochastic depth.

Motivated by the ensemble interpretation of residual networks in Veit et al. [33] and the results with stochastic depth [10], we devised another method to handle the phenomenon associated with deleting the downsampling unit. In the proposed method, the feature map dimensions are increased at all layers to distribute the burden concentrated at locations of residual units affected by downsampling, such that it is equally distributed across all units. It was found that using the proposed new network architecture, deleting the units with downsampling does not degrade the performance significantly. In our paper, we refer to this network architecture as a deep "pyramidal" network and a "pyramidal" residual network with a residual-type network architecture. This reflects the fact that the shape of the network architecture can be compared to that of a pyramid. That is, the number of channels gradually increases as a function of the depth at which the layer occurs, which is similar to a pyramid structure of which the shape gradually widens from the top downwards. This structure is illustrated in comparison to other network architectures in Figure 1. The key contributions are summarized as follows:

- A deep pyramidal residual network (PyramidNet) is introduced. The key idea is to concentrate on the feature map dimension by increasing it gradually instead of by increasing it sharply at each residual unit with downsampling. In addition, our network architecture works as a mixture of both plain and residual networks by using zero-padded shortcut connections when increasing the feature map dimension.

- A novel residual unit is also proposed, which can further improve the performance of ResNet-based architectures (compared with state-of-the-art network architectures).

The remainder of this paper is organized as follows. Section 2 presents our PyramidNets and introduces a novel residual unit that can further improve ResNet. Section 3 closely analyzes our PyramidNets via several discussions. Section 4 presents experimental results and comparisons with several state-of-the-art deep network architectures. Section 5 concludes our paper with suggestions for future works.

2. Network Architecture

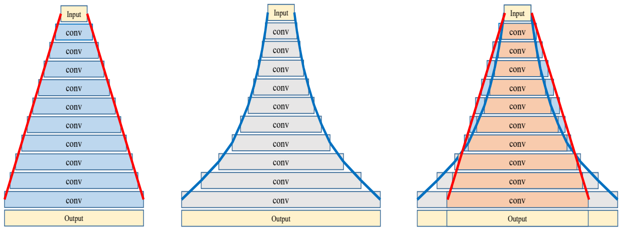
In this section, we introduce the network architectures of our PyramidNets. The major difference between PyramidNets and other network architectures is that the dimension of channels gradually increases, instead of maintaining the dimension until a residual unit with downsampling appears. A schematic illustration is shown in Figure 1 (d) to facilitate understanding of our network architecture.

2.1. Feature Map Dimension Configuration

Most deep CNN architectures [7, 8, 13, 25, 31, 35] utilize an approach whereby feature map dimensions are increased by a large margin when the size of the feature map decreases, and feature map dimensions are not increased until they encounter a layer with downsampling. In the case of the original ResNet for CIFAR datasets [12], the number of feature map dimensions D_k of the k-th residual unit that belongs to the n-th group can be described as follows:

$$D_k = \begin{cases} 16, & \text{if } n(k) = 1, \\ 16 \cdot 2^{n(k)-2}, & \text{if } n(k) \geq 2, \end{cases} \tag{1}$$

in which $n(k) \in \{1, 2, 3, 4\}$ denotes the index of the group to which the k-th residual unit belongs. The residual units that belong to the same group have an equal feature map size, and the n-th group contains N_n residual units. In the



(a) (b) (c)

Figure 2. Visual illustrations of (a) additive PyramidNet, (b) multiplicative PyramidNet, and (c) a comparison of (a) and (b).

first group, there is only one convolutional layer that converts an RGB image into multiple feature maps. For the n-th group, after N_n residual units have passed, the feature size is downsampled by half and the number of dimensions is doubled. We propose a method of increasing the feature map dimension as follows:

$$D_k = \begin{cases} 16, & \text{if } k = 1, \\ \lfloor D_{k-1} + \alpha/N \rfloor, & \text{if } 2 \le k \le N+1, \end{cases} \tag{2}$$

in which N denotes the total number of residual units, defined as $N = \sum_{n=2}^{4} N_n$. The dimension is increased by a step factor of α/N, and the output dimension of the final unit of each group becomes $16 + (n-1)\alpha/3$ with same number of residual units in each group. The details of our network architecture are presented in Table 1.

The above equations are based on an addition-based widening step factor α for increasing dimensions. However, of course, multiplication-based widening (i.e., the process of multiplying by a factor to increase the channel dimension geometrically) presents another possibility for creating a pyramid-like structure. Then, eq.(2) can be transformed as follows:

$$D_k = \begin{cases} 16, & \text{if } k = 1, \\ \lfloor D_{k-1} \cdot \alpha^{\frac{1}{N}} \rfloor, & \text{if } 2 \le k \le N+1. \end{cases} \tag{3}$$

The main difference between additive and multiplicative PyramidNets is that the feature map dimension of an additive network gradually increases linearly, whereas the dimension of a multiplicative network increases geometrically. That is, the dimension slowly increases in input-side layers and sharply increases in output-side layers. This process is similar to that of the original deep network architectures such as VGG [25] and ResNet [7]. The visual illustrations of the additive and multiplicative PyramidNets are shown in Figure 2. In this paper, we compare the performance of both of these dimension-increasing approaches by comparing an additive PyramidNet (eq. (2)) and a multiplicative PyramidNet (eq. (3)) in section 4.

Group	Output size	Building Block
conv 1	32×32	$[3 \times 3, 16]$
conv 2	32×32	$\begin{bmatrix} 3 \times 3, \lfloor 16 + \alpha(k-1)/N \rfloor \\ 3 \times 3, \lfloor 16 + \alpha(k-1)/N \rfloor \end{bmatrix} \times N_2$
conv 3	16×16	$\begin{bmatrix} 3 \times 3, \lfloor 16 + \alpha(k-1)/N \rfloor \\ 3 \times 3, \lfloor 16 + \alpha(k-1)/N \rfloor \end{bmatrix} \times N_3$
conv 4	8×8	$\begin{bmatrix} 3 \times 3, \lfloor 16 + \alpha(k-1)/N \rfloor \\ 3 \times 3, \lfloor 16 + \alpha(k-1)/N \rfloor \end{bmatrix} \times N_4$
avg pool	1×1	$[8 \times 8, 16 + \alpha]$

Table 1. Structure of our PyramidNet for benchmarking with CIFAR-10 and CIFAR-100 datasets. α denotes the widening factor, and N_n signifies the number of blocks in a group. Downsampling is performed at conv3_1 and conv4_1 with a stride of 2.

2.2. Building Block

The building block (i.e., the convolutional filter stacks with ReLUs and BN layers) in a residual unit is the core of ResNet-based architectures. It is obvious that in order to maximize the capability of the network architecture, designing a good building block is essential. As shown in Figure 6, the layers can be stacked in various manners to construct a single building block. We found the building block shown in Figure 6 (d) to be the most promising, and therefore we included this structure as building block in our PyramidNets. The discussion of this matter is continued in the following section.

In terms of shortcut connections, many researchers either use those based on identity mapping, or those employing convolution-based projection. However, as the feature map dimension of PyramidNet is increased at every unit, we can only consider two options: zero-padded identity-mapping shortcuts, and projection shortcuts conducted by 1×1 convolutions. However, as mentioned in the work of He *et al.* [8], the 1×1 convolutional shortcut produces a poor result when there are too many residual units, i.e., this shortcut is unsuitable for very deep network architectures. Therefore, we select zero-padded identity-mapping shortcuts for all residual units. Further discussions about the zero-padded shortcut are provided in the following section.

3. Discussions

In this section, we present an in-depth study of the architecture of our PyramidNet, together with the proposed novel residual units. The experiments we include here support the study and confirm that insights obtained from our network architecture can further improve the performance of existing ResNet-based architectures.

3.1. Effect of PyramidNet

According to the work of Veit *et al.* [33], ResNets can be viewed as ensembles of relatively shallow networks, supported by the observation that deleting an individual building block in a residual unit of ResNets incurs minor classi-

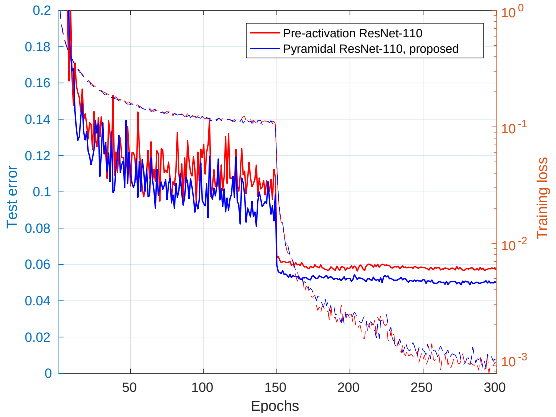


Figure 3. Performance comparison between the pre-activation ResNet [8] and our PyramidNet, using CIFAR datasets. Dashed and solid lines denote the training loss and test error, respectively.

fication loss, whereas removing layers from plain networks such as VGG [25] severely reduces the classification rate. However, in both original and pre-activation ResNets [7, 8], another noteworthy aspect is that deleting the units with downsampling (and doubling the feature dimension) still degrades performance by a large margin [33]. Meanwhile, when a stochastic depth [10] is applied, this phenomenon is not observed, and the performance is also improved, according to the experiment of Veit *et al.* [33]. The objective of our PyramidNet is to resolve this phenomenon differently, by attempting to gradually increase the feature map dimension instead of doubling it at one of the residual units and to evenly distribute the burden of increasing the feature maps. We observed that our PyramidNet indeed resolves this phenomenon and at the same time improves overall performance. We further analyze the effect of our PyramidNet by comparing it against the pre-activation ResNet, with the following experimental results.

First, we compare the training and test error curves of our PyramidNet with those of the pre-activation ResNet [8] in Figure 3. The standard pre-activation ResNet with 110 layers is used for comparison. For our PyramidNet, we used a depth of 110 layers with a widening factor of $\alpha = 48$; it had the same number of parameters (1.7M) as the pre-activation ResNet to allow for a fair comparison. The results indicate that our PyramidNet has superior test accuracy, thereby confirming its greater ability to generalize compared to existing deep networks.

Second, we verify the ensemble effect of our PyramidNets by evaluating the performance after deleting individual units, similar to the experiment of Veit *et al.* [33]. The results are shown in Figure 4. As mentioned by Veit *et al.* [33], removing individual units only causes a slight performance loss, compared with a plain network such as the VGG [25]. However, in the case of the pre-activation ResNet, removing the blocks subjected to downsampling tends to affect the classification accuracy by a relatively large margin, whereas this does not occur with our PyramidNets. Furthermore, the mean average error differences between the baseline result and the result obtained when individual units were deleted from both the pre-activation ResNet and our PyramidNet were 0.72% and 0.54%, respectively. This result shows that the ensemble effect of our PyramidNet becomes stronger than the original ResNet, such that generalization ability is improved.

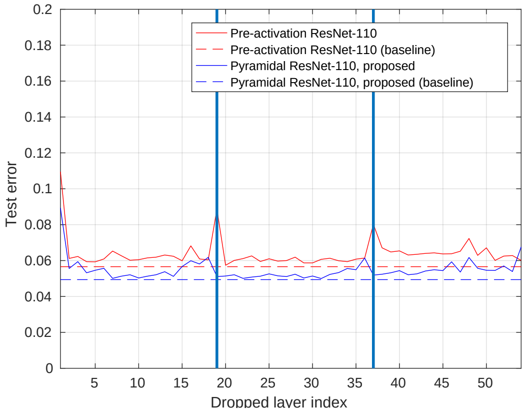


Figure 4. Test error curves to study the extent to which units contribute to the performance in different network architectures. Deletion of individual building blocks in two network architectures (our PyramidNet and the pre-activation ResNet [8]) enables us to determine which residual units contribute most to their performance. The dashed and solid lines denote the test errors that occur when no units are deleted, and when an individual unit is deleted, respectively. Bold vertical lines denote the location of residual units through downsampling.

3.2. Zero-padded Shortcut Connection

ResNets and pre-activation ResNets [7, 8] were studied several types of shortcut, such as an identity-mapping shortcut or projection shortcut. The experimental results in [8] showed that the identity-mapping shortcut connection is a much more appropriate choice than other shortcuts. Because an identity-mapping shortcut connection does not have parameters, it has a lower possibility of overfitting compared to the other shortcut connection types; this ensures improved generalization ability. Moreover, it can purely pass through the gradient according to the identity mapping, and therefore it provides more stability in the training stage.

In the case of our PyramidNet, identity mapping alone cannot be used for a shortcut because the feature map dimension differs among individual residual units. Therefore, only a zero-padded shortcut or projection shortcut can be used for all the residual units. However, as discussed in [8], a projection shortcut can hamper information propagation

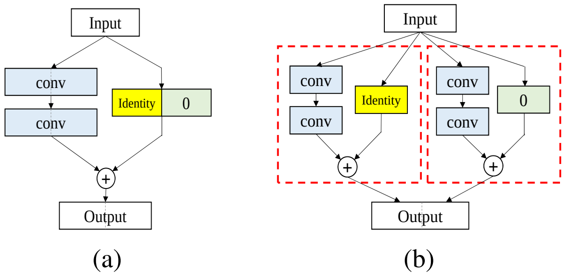


Figure 5. Structure of residual unit (a) with zero-padded identity-mapping shortcut, (b) unraveled view of (a) showing that the zero-padded identity-mapping shortcut constitutes a mixture of a residual network with a shortcut connection and a plain network.

and lead to optimization problems, especially for very deep networks. On the other hand, we found that the zero-padded shortcut does not lead to the overfitting problem because no additional parameters exist, and surprisingly, it shows significant generalization ability compared to other shortcuts.

We now examine the effect of the zero-padded identity-mapping shortcut on the k-th residual unit that belongs to the n-th group with the reshaped vector $\mathbf{x}_k^l$ of the l-th feature map:

$$\mathbf{x}_k^l = \begin{cases} \mathbf{F}_{(k,l)}(\mathbf{x}_{k-1}^l) + \mathbf{x}_{k-1}^l, & \text{if } 1 \leq l \leq D_{k-1} \\ \mathbf{F}_{(k,l)}(\mathbf{x}_{k-1}^l), & \text{if } D_{k-1} < l \leq D_k \end{cases} \tag{4}$$

where D_k represents the pre-defined channel dimensions of the k-th residual unit. From eq.(4), zero-padded elements of the identity-mapping shortcut for increasing dimension let $\mathbf{x}_k^l$ contain the outputs of both residual networks and plain networks. Therefore, we could conjecture that each zero-padded identity-mapping shortcut can provide a mixture of the residual network and plain network, as shown in Figure 5. Furthermore, our PyramidNet increases the channel dimension at every residual unit, and the mixture effect of the residual network and plain network increases markedly. Figure 4 supports the conclusion that the test error of PyramidNet does not oscillate as much as that of the pre-activation ResNet.

3.3. A New Building Block

To maximize the capability of the network, it is natural to ask the following question: **"Can we design a better building block by altering the stacked elements inside the building block in more principled way?"**. The first building block types were proposed in the original paper on ResNets [7], and another type of building block was subsequently proposed in the paper on pre-activation ResNets [8], to answer the question. Moreover, pre-activation ResNets attempted to solve the backward gradient flowing problem [8] by redesigning residual modules; this proved to be successful in trials. However, although the pre-activation residual unit was discovered with empirically improved performance, further investigation over the possible combinations is not yet performed, leaving a potential room for improvement. We next attempt to answer the question from two points of view by considering Rectified Linear Units (ReLUs) [20] and Batch Normalization (BN) [11] layers.

3.3.1 ReLUs in a Building Block

Including ReLUs [20] in the building blocks of residual units is essential for nonlinearity; however, we found empirically that the performance can vary depending on the locations and the number of ReLUs. This could be discussed with original ResNets [7], for which it was shown that the performance increases as the network becomes deeper; however, if the depth exceeds 1,000 layers, overfitting still occurs and the result is less accurate than that generated by shallower ResNets.

First, we note that using ReLUs after the addition of residual units adversely affects performance:

$$\mathbf{x}_k^l = \mathit{ReLU}(\mathbf{F}_{(k,l)}(\mathbf{x}_{k-\mathit{1}}^l) + \mathbf{x}_{k-\mathit{1}}^l), \tag{5}$$

where the ReLUs seem to have the function of filtering non-negative elements. Gross and Wilber [5] found that simply removing ReLUs from the original ResNet [7] after each addition with the shortcut connection leads to small performance improvements. This could be understood by considering that, after addition, ReLUs provide non-negative input to the subsequent residual units, and therefore the shortcut connection is always non-negative and the convolution layers would take responsibility for producing negative output before addition; this may decrease the overall capability of the network architecture as analyzed in [8]. The pre-activation ResNets proposed by He *et al.* [8] also overcame this issue with pre-activated residual units that place the BN layers and ReLUs before (instead of after) the convolution layers:

$$\mathbf{x}_k^l = \mathbf{F}_{(k,l)}(\mathbf{x}_{k-1}^l) + \mathbf{x}_{k-1}^l, \tag{6}$$

where ReLUs are removed after addition to create an identity path. Consequently, the overall performance has increased by a large margin without overfitting, even at depths exceeding 1,000 layers. Furthermore, Shen *et al.* [24] proposed a weighted residual network architecture, which locates a ReLU inside a residual unit (instead of locating ReLU after addition) to create an identity path, and showed that this structure also does not overfit even at depths of more than 1,000 layers.

Second, we found that the use of a large number of ReLUs in the blocks of each residual unit may negatively affect performance. The use of a single ReLU in the blocks of each residual unit, as shown in Figure 6 (b) and (d), was found to enhance performance compared with the use of

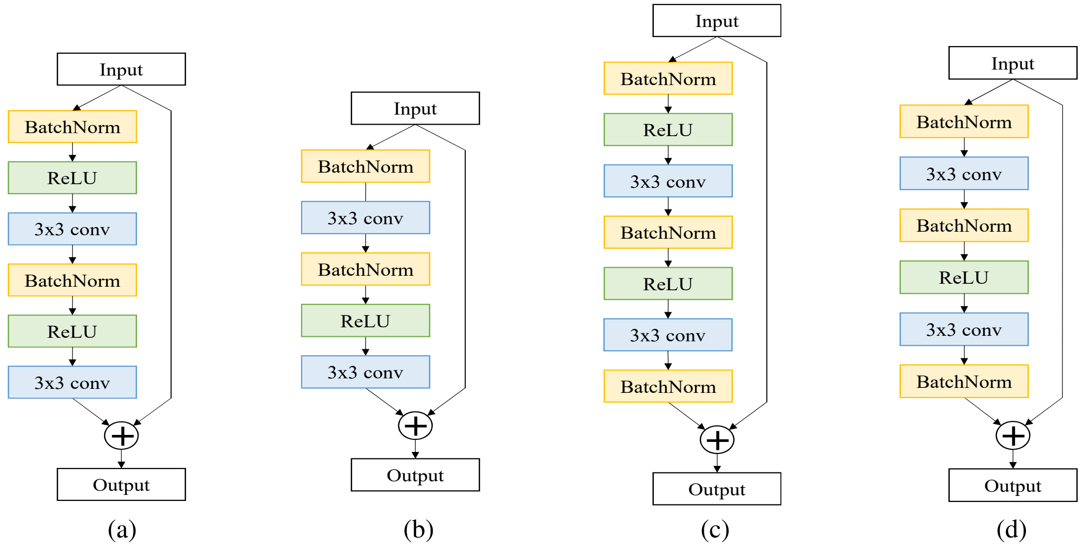


Figure 6. Various types of residual units. "BatchNorm" denotes the batch normalization layer. (a) original pre-activation ResNet [8], (b) single-ReLU pre-activation ResNet, (c) pre-activation ResNet with Batch Normalization (BN) after convolution layer, and (d) single-ReLU pre-activation ResNet with BN after convolution.

two ReLUs in the blocks shown in Figure 6 (a) and (c). Experimentally, we found that removal of the first ReLU in the stack is preferable and that the other ReLU should remain to ensure nonlinearity. Removal of the second ReLU in Figure 6 (a) changes the blocks to *BN-ReLU-conv-BN-conv*, and it is clear that, in these blocks, the convolutional layers are successively located without ReLUs to weaken their representation powers of each other. However, when we remove the first ReLU, the blocks are changed to *BN-conv-BN-ReLU-conv*, in which case the two convolution layers are separated by the second ReLU, thereby guaranteeing nonlinearity. The results in Table 2 confirm that the use of a single ReLU, achieved by removing the first ReLU as in (c) and (d) in Figure 6, enhances the performance. Consequently, provided that an appropriate number of ReLUs are used to guarantee the nonlinearity of the feature space manifold, the remaining ReLUs could be removed to improve network performance.

3.3.2 BN Layers in a Building Block

The main role of a BN layer is to normalize the activations for fast convergence and to improve performance. The experimental results of the four structures provided in Table 2 show that the BN layer can be used to maximize the capability of a single residual unit. A BN layer conducts an affine transformation with the following equation:

$$\mathbf{y} = \gamma \mathbf{x} + \beta, \tag{7}$$

where γ and β are learned for every activation in feature maps. We experimentally found that the learned γ and β could closely approximate 0. This implies that if the learned γ and β are both close to 0, then the corresponding activation is considered not to be useful. Weighted ResNets [24], in which the learnable weights occur at the end of their building blocks, are also similarly learned to determine whether the corresponding residual unit is useful. Thus, BN layers at the end of each residual unit are a generalized version including [24] to enable decisions to be made as to whether each residual unit is helpful. Therefore, the degrees of freedom obtained by involving γ and β from the BN layers could improve the capability of the network architecture. The results in Table 2 support the conclusion that adding a BN layer at the end of each building block, as in type (c) and (d) in Figure 6, improves the performance. Note that the aforementioned single-ReLU network is also improved by adding a BN layer after convolution. Furthermore, the results in Table 2 show that both PyramidNet and

ResNet architecture	CIFAR-10	CIFAR-100
(a) Pre-activation [8]	5.82%	25.06%
(b) Single-ReLU	5.31%	24.55%
(c) BN after conv	5.74%	24.54%
(d) BN after conv, and single-ReLU	5.29%	23.74%
PyramidNet architecture	CIFAR-10	CIFAR-100
(a) Pre-activation [8]	5.15%	24.40%
(b) Single-ReLU	4.81%	23.43%
(c) BN after conv	4.96%	23.89%
(d) BN after conv, and single-ReLU	**4.62%**	**23.31%**

Table 2. Top-1 errors on CIFAR datasets for several building block combinations of ReLUs and BN layers, using ResNet [8] (using original feature map dimension configuration) and our PyramidNet shown in Figure 6 (a)–(d).

Network	# of params	Output feat. dim.	Depth	CIFAR-10	CIFAR-100
NiN [18]	-	-	-	8.81%	35.68%
All-CNN [27]	-	-	-	7.25%	33.71%
DSN [17]	-	-	-	7.97%	34.57%
FitNet [21]	-	-	-	8.39%	35.04%
Highway [29]	-	-	-	7.72%	32.39%
Fractional Max-pooling [4]	-	-	-	4.50%	27.62%
ELU [29]	-	-	-	6.55%	24.28%
ResNet [7]	1.7M	64	110	6.43%	25.16%
ResNet [7]	10.2M	64	1001	-	27.82%
ResNet [7]	19.4M	64	1202	7.93%	-
Pre-activation ResNet [8]	1.7M	64	164	5.46%	24.33%
Pre-activation ResNet [8]	10.2M	64	1001	4.62%	22.71%
Stochastic Depth [10]	1.7M	64	110	5.23%	24.58%
Stochastic Depth [10]	10.2M	64	1202	4.91%	-
FractalNet [14]	38.6M	1,024	21	4.60%	23.73%
SwapOut v2 (width$\times$4) [26]	7.4M	256	32	4.76%	22.72%
Wide ResNet (width$\times$4) [34]	8.7M	256	40	4.97%	22.89%
Wide ResNet (width$\times$10) [34]	36.5M	640	28	4.17%	20.50%
Weighted ResNet [24]	19.1M	64	1192	5.10%	-
DenseNet [9]	27.2M	2,320	100	**3.74%**	19.25%
PyramidNet (mul, $\alpha = 4.75$)	1.7M	76	110	4.62%	23.16%
PyramidNet (add, $\alpha = 48$)	1.7M	64	110	4.62%	23.31%
PyramidNet (mul, $\alpha = 8$)	3.8M	128	110	4.50%	20.94%
PyramidNet (add, $\alpha = 84$)	3.8M	100	110	4.27%	20.21%
PyramidNet (mul, $\alpha = 27$)	28.3M	432	110	4.06%	18.79%
PyramidNet (add, $\alpha = 270$)	28.3M	286	110	**3.77%**	**18.29%**

Table 3. Top-1 error rates on CIFAR-10 and CIFAR-100 datasets. α denotes the widening factor; "add" and "mul" denote the results obtained with additive and multiplicative PyramidNets, respectively. "Output feat. dim." denotes the feature dimension of just before the last softmax classifier.

a new building block improve the performance significantly.

4. Experimental Results

We evaluate and compare the performance of our algorithm with that of existing algorithms [7, 8, 18, 24, 34] using representative benchmark datasets: CIFAR-10 and CIFAR-100 [12]. CIFAR-10 and CIFAR-100 each contain 32$\times$32-pixel color images, consists of 50,000 training images and 10,000 testing images. But in case of CIFAR-10, it includes 10 classes, and CIFAR-100 includes 100 classes. The standard data augmentation, horizontal flipping, and translation by 4 pixels are adopted in our experiments, following the common practice [18]. The results achieved by PyramidNets are based on the proposed residual unit: BN after conv, and single-ReLU as in Figure 6 (d). Our code is built on the Torch open source deep learning framework [1].

4.1. Training Settings

Our PyramidNets are trained using backpropagation [15] by Stochastic Gradient Descent (SGD) with Nesterov momentum for 300 epochs using the CIFAR-10 and CIFAR-100 datasets. The initial learning rate is set to 0.5, and is decayed by a factor of 0.1 at 150 and 225 epochs, respectively. The filter parameters are initialized by "msra" [6]. We use a weight decay of 0.0001, a dampening of 0, a momentum of 0.9, and a batch size of 128.

4.2. Performance Evaluation

In our work, we mainly use the top-1 error rate for evaluating our network architecture. This error rate is provided in Table 3 for our algorithm and the state-of-the-art algorithms. The experimental results show that our network has superior generalization ability, in terms of the number of parameters, showing the best results (smallest error) compared with other methods. For $\alpha = 48$, we found that the additive PyramidNet outperforms other methods such as ResNets and pre-activation Resnets with the same number of parameters (1.7M).

The experimental results show that the performance of both additive and multiplicative PyramidNets is superior to other state-of-the-art methods. When the number of parameters is low, both additive and multiplicative PyramidNets show similar performance, because these two network architectures do not have significant structural differences. However, as the number of parameters increases, they start to show a more marked difference in terms of the feature map dimension configuration. In the case of additive PyramidNets, because the feature map dimension increases linearly, the feature map dimensions of the input-side layers

Network	# of params	Output feat. dim.	Augmentation	Train crop	Test Crop	Top-1	Top-5
ResNet-152 [7]	60.0M	2,048	scale	224×224	224×224	23.0	6.7
Pre-ResNet-152† [8]	60.0M	2,048	scale+asp ratio	224×224	224×224	22.2	6.2
Pre-ResNet-200† [8]	64.5M	2,048	scale+asp ratio	224×224	224×224	21.7	5.8
PyramidNet-200 ($\alpha = 300$)	62.1M	1,456	scale+asp ratio	224×224	224×224	**20.5**	**5.3**
PyramidNet-200 ($\alpha = 450$)	116.4M	2,056	scale+asp ratio	224×224	224×224	**20.1**	**5.4**
ResNet-200 [7]	64.5M	2,048	scale	224×224	320×320	21.8	6.0
Pre-ResNet-200 [8]	64.5M	2,048	scale+asp ratio	224×224	320×320	20.1	4.8
Inception-v3 [32]	-	2,048	scale+asp ratio	299×299	299×299	21.2	5.6
Inception-ResNet-v1 [30]	-	1,792	scale+asp ratio	299×299	299×299	21.3	5.5
Inception-v4 [30]	-	1,536	scale+asp ratio	299×299	299×299	20.0	5.0
Inception-ResNet-v2 [30]	-	1,792	scale+asp ratio	299×299	299×299	19.9	4.9
PyramidNet-200 ($\alpha = 300$)	62.1M	1,456	scale+asp ratio	224×224	320×320	**19.6**	**4.8**
PyramidNet-200 ($\alpha = 450$)	116.4M	2,056	scale+asp ratio	224×224	320×320	**19.2**	**4.7**

Table 4. Comparisons of single-crop error on the ILSVRC 2012 validation set. We use the additive PyramidNet for our model. "asp ratio" means the aspect ratio applied for data augmention, and "Output feat. dim." denotes the feature dimension of just after the last global pooling layer. † denotes the results obtained from *https://github.com/facebook/fb.resnet.torch*.

tend to be larger, and those of the output-side layers tend to be smaller, compared with multiplicative PyramidNets as illustrated in Figure 2 (c).

Typically, researchers [7, 25] use multiplicative scaling of feature map dimension for downsampling modules, which is implemented to give more degrees of freedom to the final layer (i.e., the classification part) by increasing the feature map dimension of the output-side layers. From the experimental results, the additive PyramidNets show improved performance compared to the multiplicative PyramidNets. These results imply that the input-side layers, which have large feature maps, play a much more important role than the output-side layers do; this indicates that increasing the model capacity of the input-side layers would lead to a better performance improvement than increasing the model capacity of the output-side layers.

We also note that, although the use of regularization methods such as dropout [28] or stochastic depth [10] could further improve the performance of our method, we did not involve those methods to ensure a fair comparison with other methods.

4.3. ImageNet

1,000-class ImageNet dataset [22] used for ILSVRC contains more than one million training images and 50,000 validation images. We use the additive PyramidNets for the experiment. Like other methods [7, 8], we use the bottleneck architecture for our PyramidNet, deleting the first ReLU layer and adding a BN layer at the last layer as described in Section 3.3 for further performance improvement. Thus, the residual unit of our PyramidNet consists of *BN-conv(1×1)-BN-ReLU-conv(3×3)-BN-ReLU-conv(1×1)-BN*, with a zero-padded shortcut.

We train our models for 120 epochs with a batch size of 128, and the initial learning rate is set to 0.05, divided by 10 at 60, 90 and 105 epochs. We use the same weight decay, momentum, and initialization settings as those of CIFAR datasets. We train our model by using a standard data augmentation with scale jittering and aspect ratio as suggested in Szegedy *et al.* [31]. Table 4 shows the results of our PyramidNets in ImageNet dataset compared with the state-of-the-art methods. In case of the result with $\alpha = 450$, we use dropout [28] method after the global average pooling layer for regularization, and for $\alpha = 300$, we do not use a dropout for a fair comparison with other methods. The experimental results show that our PyramidNet with $\alpha = 300$ has a top-1 error rate of 20.5%, which is 1.2% lower than the pre-activation ResNet-200 [8] which has a similar number of parameters but higher output feature dimension than our model. We also notice that increasing α with an appropriate regularization method can further improve the performance.

For comparison with the Inception-ResNet [30] that uses a testing crop with 299×299 size, we test our model on a 320×320 crop, by the same reason with the work of He *et al.* [8]. Our PyramidNet with $\alpha = 300$ shows a top-1 error rate of 19.6%, which outperforms both the pre-activation ResNet [8] and the Inception-ResNet-v2 [30] models.

5. Conclusion

The main idea of the novel deep network architecture described in this paper involves increasing the feature map dimension gradually, in order to construct so-called PyramidNets along with the concept of ResNets. We also developed a novel residual unit, which includes a new building block for a residual unit with a zero-padded shortcut; this design leads to significantly improved generalization ability. In tests using the CIFAR-10 and CIFAR-100 datasets, our PyramidNets outperform all previous state-of-the-art deep network architectures. Furthermore, the insights in this paper could be utilized by any network architecture, to improve their capacity for better performance. In future work, we will develop methods of optimizing parameters such as feature map dimensions in more principled ways with proper cost functions that give insight into the nature of residual networks.

References

[1] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011. 7

[2] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014. 1

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1

[4] B. Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014. 7

[5] S. Gross and M. Wilber. Training and investigating residual nets. 2016. http://torch.ch/blog/2016/02/04/resnets.html. 5

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 7

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 3, 4, 5, 7, 8

[8] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 1, 2, 3, 4, 5, 6, 7, 8

[9] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. 7

[10] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 1, 2, 4, 7, 8

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 5

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. In *Tech Report*, 2009. 2, 7

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012. 1, 2

[14] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016. 7

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 7

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[17] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015. 7

[18] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014. 7

[19] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1

[20] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 5

[21] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015. 7

[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. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1, 8

[23] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014. 1

[24] F. Shen and G. Zeng. Weighted residuals for very deep networks. *arXiv preprint arXiv:1605.08831*, 2016. 5, 6, 7

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2, 3, 4, 8

[26] S. Singh, D. Hoiem, and D. Forsyth. Swapout: Learning an ensemble of deep architectures. In *NIPS*, 2016. 7

[27] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR Workshop*, 2015. 7

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 8

[29] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *NIPS*, 2015. 1, 7

[30] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. In *ICLR Workshop*, 2016. 1, 8

[31] 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. 1, 2, 8

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 8

[33] A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, 2016. 1, 2, 3, 4

[34] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016. 2, 7

[35] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 1, 2