Shakedrop Regularization for Deep Residual Learning

YOSHIHIRO YAMADA[1], **MASAKAZU IWAMURA**[1], **(Member, IEEE),**
TAKUYA AKIBA[2], **AND KOICHI KISE**[1], **(Member, IEEE)**

[1]Graduate School of Engineering, Osaka Prefecture University, Osaka 599-8531, Japan
[2]Preferred Networks, Inc., Chiyoda Tokyo 100-0004, Japan

Corresponding author: Yoshihiro Yamada (yamada@m.cs.osakafu-u.ac.jp)

ABSTRACT Overfitting is a crucial problem in deep neural networks, even in the latest network architectures. In this paper, to relieve the overfitting effect of ResNet and its improvements (i.e., Wide ResNet, PyramidNet, and ResNeXt), we propose a new regularization method called ShakeDrop regularization. ShakeDrop is inspired by Shake-Shake, which is an effective regularization method, but can be applied to ResNeXt only. ShakeDrop is more effective than Shake-Shake and can be applied not only to ResNeXt but also ResNet, Wide ResNet, and PyramidNet. An important key is to achieve stability of training. Because effective regularization often causes unstable training, we introduce a training stabilizer, which is an unusual use of an existing regularizer. Through experiments under various conditions, we demonstrate the conditions under which ShakeDrop works well.

INDEX TERMS Computer vision, image classification, neural networks.

I. INTRODUCTION

Recent advances in generic object recognition have been achieved using deep neural networks. Since ResNet [11] created the opportunity to use very deep convolutional neural networks (CNNs) of over a hundred layers by introducing the building block, its improvements, such as Wide ResNet [36], PyramidNet [8], [9], and ResNeXt [33] have broken records for the lowest error rates.

The development of such base network architectures, however, is not sufficient to reduce the generalization error (i.e., difference between the training and test errors) due to overfitting. In order to improve test errors, regularization methods which are processes to introduce additional information to CNNs have been proposed [23]. Widely used regularization methods include data augmentation [18], stochastic gradient descent (SGD) [37], weight decay [20], batch normalization (BN) [15], label smoothing [28], adversarial training [6], mixup [29]–[31], [38], and dropout [26], [32]. Because the generalization errors when regularization methods are used

are still large, effective regularization methods have been studied.

Recently, an effective regularization method which achieved the lowest test error called Shake-Shake regularization [4], [5] was proposed. It is an interesting method, which, in training, disturbs the calculation of the forward pass using a random variable, and also that of the backward pass using a different random variable. Its effectiveness was proven by an experiment on ResNeXt, to which Shake-Shake was applied (hereafter, this type of combination is denoted by ''ResNeXt + Shake-Shake''), which achieved the lowest error rate on CIFAR-10/100 datasets [17]. Shake-Shake, however, has the following two drawbacks: (i) it can be applied to ResNeXt only, and (ii) the reason it is effective has not yet been identified.

The current paper addresses these problems. For problem (i), we propose a novel powerful regularization method called *ShakeDrop regularization*, which is more effective than Shake-Shake. Its main advantage is that it has the potential to be applied not only to ResNeXt (hereafter, three-branch architectures) but also ResNet, Wide ResNet, and PyramidNet (hereafter, two-branch architectures). The main difficulty to overcome is unstable training. We solve

---

The associate editor coordinating the review of this manuscript and approving it for publication was Huanqiang Zeng.
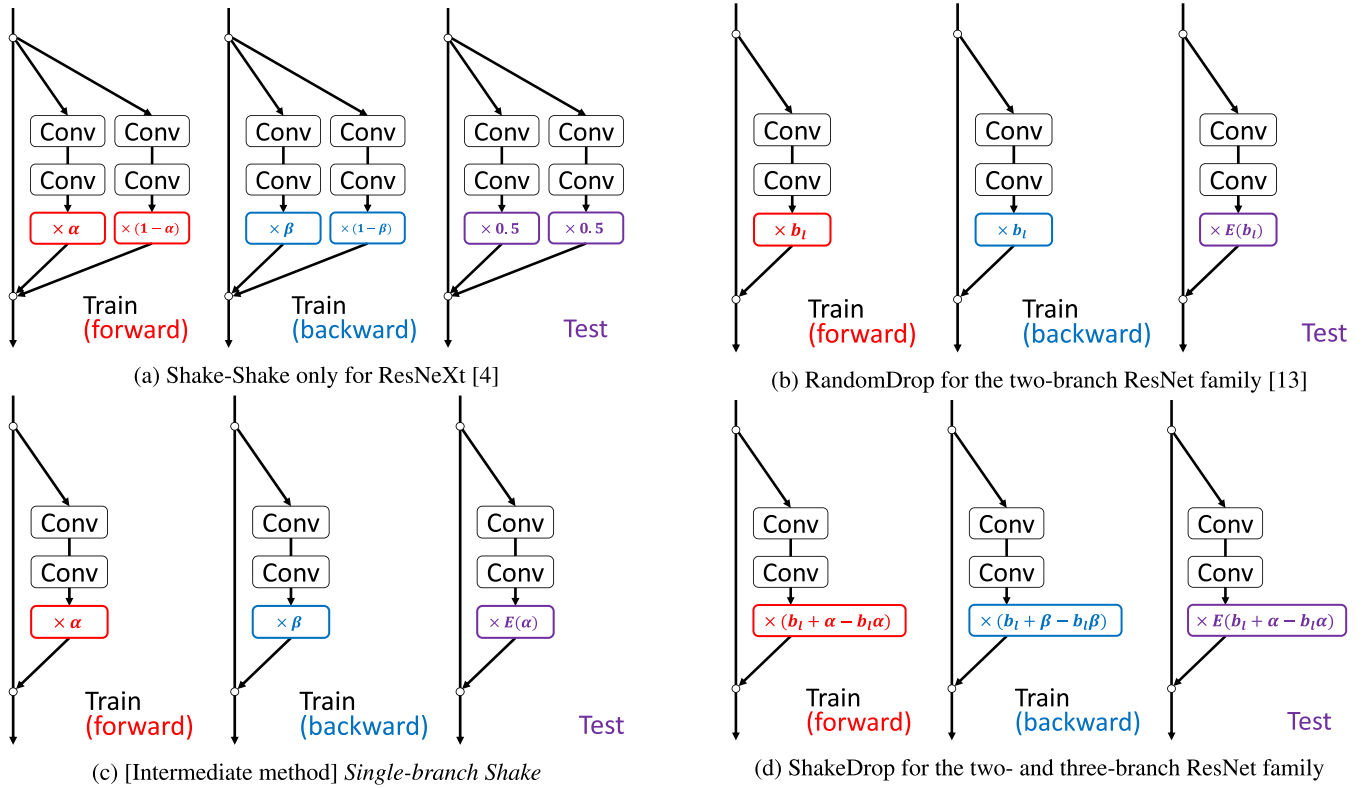
FIGURE 1. Regularization methods for the ResNet family. (a) and (b) are existing methods. (c) is an intermediate regularization method used to derive the proposed method. (d) is the proposed method. "Conv" denotes a convolution layer; $E[x]$ denotes the expected value of x; and α, β, and b_l denote random coefficients.

this problem by proposing a new stabilizing mechanism for *difficult-to-train* networks. For problem (ii), in the process of deriving ShakeDrop, we provide an intuitive interpretation of Shake-Shake. Additionally, we present the mechanism in which ShakeDrop works. Through experiments using various base network architectures and parameters, we demonstrate the conditions under which ShakeDrop successfully works.

This paper is an extended version of ICLR workshop paper [35].

II. REGULARIZATION METHODS FOR THE RESNET FAMILY

In this section, we present two regularization methods for the ResNet family, both of which are used to derive the proposed method.

Shake-Shake regularization [4], [5] is an effective regularization method for ResNeXt. It is illustrated in Fig. 1. The basic ResNeXt building block, which has a three-branch architecture, is given as

$$G(x) = x + F_1(x) + F_2(x), \quad (1)$$

where x and $G(x)$ are the input and output of the building block, respectively, and $F_1(x)$ and $F_2(x)$ are the outputs of two residual branches.

Let α and β be independent random coefficients uniformly drawn from the uniform distribution on the interval $[0, 1]$. Then Shake-Shake is given as

$$G(x) = \begin{cases} x + \alpha F_1(x) + (1-\alpha)F_2(x), & \text{in train-fwd} \\ x + \beta F_1(x) + (1-\beta)F_2(x), & \text{in train-bwd} \\ x + E[\alpha]F_1(x) + E[1-\alpha]F_2(x), & \text{in test,} \end{cases}$$

$$(2)$$

where *train-fwd* and *train-bwd* denote the forward and backward passes of training, respectively. Expected values $E[\alpha] = E[1-\alpha] = 0.5$. Equation (2) means that the calculation of the forward pass is multiplied by random coefficient α and that of the backward pass by another random coefficient β. The values of α and β are drawn for each image or batch. In this paper, we suggest training for longer than usual (more precisely, six times as long as usual).

In the training of neural networks, if the output of a residual branch is multiplied by coefficient α in the forward pass, then it is natural to multiply the gradient by the same coefficient (i.e., α) in the backward pass. Hence, compared with the standard approach, Shake-Shake makes the gradient β/α times as large as the correctly calculated gradient on one branch and $(1-\beta)/(1-\alpha)$ times on the other branch. It seems that the disturbance prevents the network parameters from being captured in local minima. However, the reason why such a disturbance is effective has not been sufficiently identified.

RandomDrop regularization (a.k.a., Stochastic Depth and ResDrop) [14] is a regularization method originally proposed for ResNet, and also applied to PyramidNet [34]. It is illustrated in Fig. 1(b). The basic ResNet building block, which has a two-branch architecture, is given as

$$G(x) = x + F(x), \qquad (3)$$

where $F(x)$ is the output of the residual branch. Random-Drop makes the network appear to be shallow in learning by dropping some stochastically selected building blocks. The lth building block from the input layer is given as

$$G(x) = \begin{cases} x + b_l F(x), & \text{in train-fwd} \\ x + b_l F(x), & \text{in train-bwd} \\ x + E[b_l]F(x), & \text{in test,} \end{cases} \qquad (4)$$

where $b_l \in \{0, 1\}$ is a Bernoulli random variable with the probability $P(b_l = 1) = E[b_l] = p_l$. In this paper, we recommend the *linear decay rule* to determine p_l, which is given as

$$p_l = 1 - \frac{l}{L}(1 - P_L), \qquad (5)$$

where L is the total number of building blocks and p_L is the initial parameter. We suggest using $p_L = 0.5$.

RandomDrop can be regarded as a simplified version of dropout [26]. The main difference is that RandomDrop drops layers, whereas dropout drops elements.

III. PROPOSED METHOD
A. SHAKEDROP REGULARIZATION
The proposed *ShakeDrop*, illustrated in Fig. 1(d), is given as

$$G(x) = \begin{cases} x + (b_l + \alpha - b_l\alpha)F(x), & \text{in train-fwd} \\ x + (b_l + \beta - b_l\beta)F(x), & \text{in train-bwd} \\ x + E[b_l + \alpha - b_l\alpha]F(x), & \text{in test,} \end{cases} \qquad (6)$$

where b_l is a Bernoulli random variable with probability $P(b_l = 1) = E[b_l] = p_l$ given by the linear decay rule (5) in each layer, and α and β are independent uniform random variables in each element. The most effective ranges of α and β were experimentally found to be different from those of Shake-Shake, and are $\alpha = 0$, $\beta \in [0, 1]$ and $\alpha \in [-1, 1]$, $\beta \in [0, 1]$. Further details of the parameters are presented in Sections IV and V.

In the training phase, b_l controls the behavior of Shake-Drop. If $b_l = 1$, then (6) is deformed as

$$G(x) = \begin{cases} x + F(x), & \text{in train-fwd} \\ x + F(x), & \text{in train-bwd;} \end{cases} \qquad (7)$$

that is, ShakeDrop is equivalent to the original network (e.g., ResNet). If $b_l = 0$, then (6) is deformed as

$$G(x) = \begin{cases} x + \alpha F(x), & \text{in train-fwd} \\ x + \beta F(x), & \text{in train-bwd;} \end{cases} \qquad (8)$$

that is, the calculation of $F(x)$ is perturbed by α and β.

B. DERIVATION OF SHAKEDROP
1) INTERPRETATION OF SHAKE-SHAKE REGULARIZATION
We provide an intuitive interpretation of Shake-Shake; to the best of our knowledge, it has not been provided yet. As shown in (2) (and in Fig. 1), in the forward pass, Shake-Shake interpolates the outputs of two residual branches (i.e., $F_1(x)$ and $F_2(x)$) with random weight α. DeVries and Taylor [2] demonstrated that the interpolation of two data in the feature space can synthesize reasonable augmented data; hence the interpolation in the forward pass of Shake-Shake can be interpreted as synthesizing reasonable augmented data. The use of random weight α enables us to generate many different augmented data. By contrast, in the backward pass, a different random weight β is used to disturb the updating parameters, which is expected to help to prevent parameters from being caught in local minima by enhancing the effect of SGD [16].

2) SINGLE-BRANCH SHAKE REGULARIZATION
The regularization mechanism of Shake-Shake relies on two or more residual branches; hence, it can only be applied to three-branch network architectures (i.e., ResNeXt). To achieve a similar regularization to Shake-Shake on two-branch architectures (i.e., ResNet, Wide ResNet, and PyramidNet), we need a different mechanism from interpolation in the forward pass that can synthesize augmented data in the feature space. In fact, DeVries and Taylor [2] demonstrated not only interpolation but also noise addition in the feature space, which generates reasonable augmented data. Hence, following Shake-Shake, we apply random perturbation to the output of a residual branch (i.e., $F(x)$ of (3)); that is, it is given as

$$G(x) = \begin{cases} x + \alpha F(x), & \text{in train-fwd} \\ x + \beta F(x), & \text{in train-bwd} \\ x + E[\alpha]F(x), & \text{in test.} \end{cases} \qquad (9)$$

We call this regularization method *Single-branch Shake*. It is illustrated in Fig. 1(c). *Single-branch Shake* is expected to be as effective as Shake-Shake. However, it does not work well in practice. For example, in our preliminary experiments, we applied it to 110-layer PyramidNet with $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ following Shake-Shake. However, the result on the CIFAR-100 dataset was significantly bad (i.e., an error rate of 77.99%).

3) STABILIZATION OF TRAINING
In this section, we consider what caused the failure of *Single-branch Shake*. A natural guess is that Shake-Shake has a stabilizing mechanism that *Single-branch Shake* does not have. The mechanism is ''two residual branches.'' We present an argument to verify whether this is the case. As presented in Section II, in training, Shake-Shake makes the gradients of two branches β/α times and $(1 - \beta)/(1 - \alpha)$ times as large as the correctly calculated gradients. Thus, when α is close to zero or one, it cannot converge (ruin) training because it could

TABLE 1. Regularization methods that generate new data. "Sample-wise generation" means that data is generated using a single sample.

Regularization method	Data augmentation			Sample-wise generation
	In (input) data space	In feature space	In label space	
Data augmentation [18]	✓			✓
Adversarial training [6]	✓			✓
Label smoothing [28]			✓	✓
Mixup [29], [30], [38]	✓		✓	
Manifold mixup [31]	✓	✓	✓	
Shake-Shake [4], [5]		✓		✓
ShakeDrop		✓		✓

make a gradient prohibitively large.[1] However, two residual branches of Shake-Shake work as a fail-safe system; that is, even if the coefficient on one branch is large, the other is kept small. Hence, training on at least one branch is not ruined. *Single-branch Shake*, however, does not have such a fail-safe system.

From the discussion above, the failure of *Single-branch Shake* was caused by the perturbation being too strong and the lack of a stabilizing mechanism. Because weakening the perturbation would just weaken the effect of regularization, we need a method to stabilize unstable learning under strong perturbation.

We propose using the mechanism of RandomDrop to solve the issue. RandomDrop is designed to make a network apparently shallow to avoid the problems of vanishing gradients, diminishing feature reuse, and a long training time. In our scenario, the original use of RandomDrop does not have a positive effect because a shallower version of a strongly perturbed network (e.g., a shallow version of "PyramidNet + *Single-branch Shake*") would also suffer from strong perturbation. Thus, we use the mechanism of RandomDrop as a probabilistic switch for the following two network architectures:

1) the original network (e.g., PyramidNet), which corresponds to (7), and
2) a network that suffers from strong perturbation (e.g., "PyramidNet + *Single-branch Shake*"), which corresponds to (8).

By mixing them up, as shown in Fig. 2, it is expected that (i) when the original network is selected, learning is correctly promoted, and (ii) when the network with strong perturbation is selected, learning is disturbed.

To achieve good performance, the two networks should be well balanced, which is controlled by parameter p_L. We discuss this issue in Section IV.

C. RELATIONSHIP WITH EXISTING REGULARIZATION METHODS

In this section, we discuss the relationship between Shake-Drop and existing regularization methods. Among them,

---

[1]This idea is supported by an experiment that limited the ranges of α and β in Shake-Shake [4]. When α and β were kept close (more precisely, on the number line, α and β were on the same side of 0.5, such as $\alpha = 0.1$ and $\beta = 0.2$), Shake-Shake achieved relatively high accuracy. However, when α and β were kept far apart (α and β were on the opposite sides of 0.5, such as $\alpha = 0.1$ and $\beta = 0.7$), the accuracy was relatively low. This indicates that when β/α or $(1 - \beta)/(1 - \alpha)$ were large, training could become less stable.
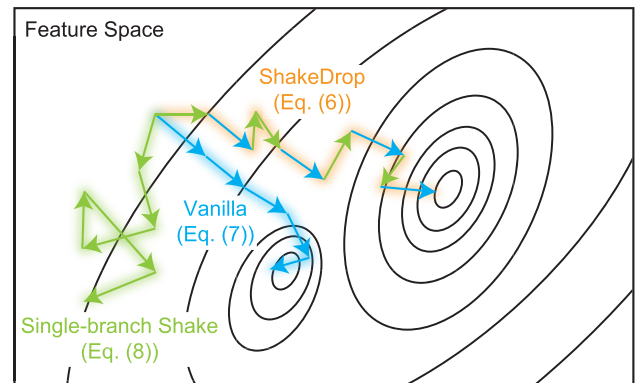

FIGURE 2. Conceptual sketch of converging trajectories. The original networks called Vanilla (7) can converge but become trapped in local minima. Single-branch Shake (8), which updates the parameters with the strong perturbation, does not become trapped in local minima but cannot converge. Using the probabilistic switches of (7) and (8), ShakeDrop is expected to not become in local minima and to converge to a better minimum.

SGD and weight decay are commonly used techniques in the training of deep neural networks. Although they were not designed for regularization, researchers have indicated that they have generalization effects [20], [37]. BN [15] is a strong regularization technique that has been widely used in recent network architectures. ShakeDrop is appended to these regularization methods.

ShakeDrop differs from RandomDrop [14] and dropout [26], [32] in the following two ways: they do not explicitly generate new data and they do not update network parameters based on noisy gradients. ShakeDrop coincides with RandomDrop when $\alpha = \beta = 0$ instead of the recommended parameters.

Some methods regularize by generating new data. They are summarized in Table 1. Data augmentation [18] and adversarial training [6] synthesize data in the (input) data space. They differ in how they generate data. The former uses manually designed means, such as random crop and horizontal flip, whereas the latter automatically generates data that should be used for training to improve generalization performance. Label smoothing [28] generates (or changes) labels for existing data. The methods mentioned above generate new data using a single sample. By contrast, some methods require multiple samples to generate new data. Mixup [38], BC learning [30], and RICAP [29] generate new data and their corresponding class labels by interpolating two or more data. Although they generate new data in the data space, manifold

TABLE 2. [Ranges of α and β] Average top-1 errors (%) of "ResNet + ShakeDrop," "ResNet (EraseReLU version) + ShakeDrop," and "PyramidNet + ShakeDrop" of four runs at the final (300th) epoch on the CIFAR-100 dataset using the batch-level update rule. "×" indicates that learning did not converge. Cases A and B are equivalent to not including the regularization method (we call this Vanilla) and RandomDrop, respectively.

Case	α	β	ResNet	ResNet (EraseReLU)	PyramidNet	Note
A	1	1	27.42	25.38	18.01	Vanilla
B	0	0	24.07	22.86	17.74	RandomDrop
C	1	0	27.95	27.57	20.87	
D	1	$[0, 1]$	26.66	25.98	18.80	
E	1	$[-1, 1]$	28.45	28.23	21.69	
F	0	1	27.15	39.09	×	
G	0	$[0, 1]$	**23.77**	**21.81**	×	
H	0	$[-1, 1]$	24.69	23.22	×	
I	$[0, 1]$	1	25.11	23.24	38.48	
J	$[0, 1]$	0	25.93	24.73	19.68	
K	$[0, 1]$	$[0, 1]$	24.78	23.75	18.27	
L	$[0, 1]$	$[-1, 1]$	26.41	25.32	20.61	
M	$[-1, 1]$	1	×	39.98	18.68	
N	$[-1, 1]$	0	24.83	23.25	**17.28**	
O	$[-1, 1]$	$[0, 1]$	×	**22.59**	**16.22**	
P	$[-1, 1]$	$[-1, 1]$	25.85	23.91	18.26	

TABLE 3. [Update rule of α and β] Average top-1 errors (%) of "PyramidNet + ShakeDrop" for four runs at the final (300th) epoch on the CIFAR-100 dataset.

α	β	Level	ResNet	PyramidNet
0	$[0, 1]$	Batch	23.77	-
		Image	99.00	
		Channel	66.30	
		Pixel	**23.74**	
$[-1, 1]$	$[0, 1]$	Batch	-	16.22
		Image		16.04
		Channel		16.12
		Pixel		**15.78**

mixup [31] also does it in the feature space. Compared with ShakeDrop, which generates data in the feature space using a single sample, none of these regularization methods are in the same category, except for Shake-Shake.

Note that the selection of regularization methods is not always exclusive. We have successfully used ShakeDrop combined with mixup (see Section V-D). Although regularization methods in the same category may not be used together (e.g., "mixup and BC learning" and "ShakeDrop and Shake-Shake"), those of different categories may be used together. Thus, developing the best method in a category is meaningful.

IV. PRELIMINARY EXPERIMENTS

ShapeDrop has three parameters: α, β, and p_L. Additionally, four possible update rules of α and β exist. In this section, we search for the best parameters of α and β and best update rule on the CIFAR-100 dataset. The best parameters found are used in the experiments in Section V. Following Random-Drop regularization [14], we used $p_L = 0.5$ as the default.

A. RANGES OF α AND β

The best parameter ranges of α and β were experimentally explored. We applied ShakeDrop to three network architectures: ResNet, ResNet (EraseReLU version), and PyramidNet. In the *EraseReLU version*, the rectified linear unit (ReLU) at the bottom of the building blocks was erased [3]. Note that EraseReLU does not affect PyramidNet

because it does not have the ReLU at the bottom of the building blocks.

Table 2 shows the representative parameter ranges of α and β that we tested and their results. Cases A and B correspond to the vanilla network (i.e., without regularization) and Ran-domDrop, respectively. On all three network architectures, case B was better than case A. We consider the results of the three network architectures individually.

- **PyramidNet** achieved the lowest error rates among the three network architectures. Only cases N and O outperformed case B. Among them, case O was the best.
- **ResNet** had a different tendency from PyramidNet: case O, which was the best on PyramidNet, did not converge. Only case G outperformed case B.
- **ResNet (EraseReLU version)** had the characteristics of both PyramidNet and ResNet; that is, both cases O and G outperformed case B. Case G was the best.

Through experiments using various base network architectures shown in Section V, we found that case O was effective on "EraseReLU"ed architectures. By contrast, case G was effective on non-"EraseReLU"ed architectures.

B. UPDATE RULE OF α AND β

The best update rule of α and β was found from the *batch*, *image*, *channel*, and *pixel* levels. ShakeDrop is determined to drop or not on each building block. Differently from Dropout and RandomDrop, even if a building block is determined to be dropped, we still have a freedom to choose how α and β are determined. That is, α and β can be drawn for each batch in parallel, each image on a batch in parallel, each channel, or each element.

In the experiment, the best α and β found in Section IV-A (i.e., $\alpha = 0$ and $\beta \in [0, 1]$ for ResNet, and $\alpha \in [-1, 1]$ and $\beta \in [0, 1]$ for PyramidNet), were used.[2] Table 3 shows that the pixel level was the best for both ResNet and PyramidNet.

---

[2]Table 3 contains cells with a hyphen ("-"). We did not conduct experiments in these settings because they were not expected to improve the error rates in the experiment, as shown in Section IV-A.

TABLE 4. [Combinations of (α, β)] Top-1 errors (%) of "PyramidNet + ShakeDrop" at the final (300th) epoch on the CIFAR-100 dataset in the batch-level update rule. Combinations of α and β used in each case are marked. Results with * are quoted from Table 2.

Case	$\alpha=1, \beta=1$	$\alpha=1, \beta=0$	$\alpha=-1, \beta=1$	$\alpha=-1, \beta=0$	PyramidNet	Note
a	✓	✓	✓	✓	**16.62**	
b		✓	✓	✓	19.51	
c	✓		✓	✓	18.79	
d	✓	✓		✓	**16.57**	
e	✓	✓	✓		**16.43**	
f		✓		✓	37.11	
g		✓		✓	17.49	
h		✓	✓		17.25	
i	✓			✓	**16.24**	
j	✓		✓		17.39	
k	✓	✓			19.18	
l				✓	25.81	
m			✓		88.75	
n		✓			*20.87	Case C in Table 2
o	✓				*18.01	Vanilla

C. COMBINATIONS OF (α, β) FOR ANALYZING SHAKEDROP BEHAVIOR

Although we successfully found effective ranges of α and β, and their update rule, we still do not understand what mechanism contributes to improving the generalization performance of ShakeDrop. One reason is that α and β are random variables. Because of this, at the end of training, we can obtain a network that is trained using various observed values of α and β. This makes it more difficult to understand the mechanism.

Hence, in this section, we explore effective combinations of (α, β). The combinations of (α, β) are defined as follows: From the best ranges of α and β for Pyramid-Net, which are $\alpha \in [-1, 1]$ and $\beta \in [0, 1]$, by taking both ends of ranges, we obtain a set of (α, β) pairs: $\{(1, 1), (1, 0), (-1, 1), (-1, 0)\}$. Then, we examine all its combinations, which are shown in Table 4. Intuitively, when $b_l = 0$, instead of drawing α and β in the ranges, a pair (α, β) is selected from its pool with equal probability.

Table 4 shows combinations of (α, β) and their results on PyramidNet. Compared with the best result in Table 2 (i.e., case O; 16.22%), the results in Table 4 are almost comparable. In particular, the best result in Table 4 (i.e., case i; 16.24%) is almost equivalent. This indicates that the random drawing of α and β in certain ranges is not the primary factor for improving error rates.

Additionally, we observe that p_L is important to error rates. As mentioned above, $(1, 1)$ is the normal state. Hence, the difference between cases i and l exists only in p_L: because case i has two elements and one of them is the normal state (i.e., $(1, 1)$), its p_L actually works as $(1 + p_L)/2$. For example, when $p_L = 0.5$, case i is equivalent to case l with $p_L = 0.75$. Cases j and m, and cases n and k have the same relationship. A comparison of their error rates shows that p_L greatly affects the error rates. We discuss this issue in Section V-E.

For further analysis, we focus on the difference among $(1, 0)$, $(-1, 1)$, and $(-1, 0)$.

What does $(\alpha, \beta) = (1, 0)$ do? (What the meaning of $\beta = 0$?)

$\alpha = 1$ indicates that the forward pass is normal. Hence, no regularization effect is expected. When $\beta = 0$, the network parameters of *the layers selected for perturbation* (i.e., the layers with $b_l = 0$) are not updated. In layers other than the selected layers, the network parameters are updated as usual. One exception is that, as the network parameters of the selected layers are not updated, other layers compensate for the amount that should be updated on the selected layers. Cases n and k contain $(1, 0)$. They were slightly worse than the best cases.

What does $(\alpha, \beta) = (-1, 1)$ do? (What the meaning of $\alpha = -1$?)

When $\alpha = -1$, in the selected layers, the calculation of the forward pass is perturbed by $\alpha = -1$. Then, the effect of perturbation is propagated to the succeeding layers. Hence, not only the selected layers but also their succeeding layers are perturbed. In the backward pass, when α is negative, the network parameters of the selected layers are updated toward the opposite direction to usual. Because of this, the network parameters of the selected layers are strongly perturbed by negative α. This can be a destructive update. In layers other than the selected layers, it is less probable that the update of the network parameters is destructive because they follow the normal update rule (equivalent to $\alpha = 1$). Cases j and m contain $(-1, 1)$. The former was slightly worse than the best and the latter was significantly bad.

What does $(\alpha, \beta) = (-1, 0)$ do?

As this is a combination of $\alpha = -1$ and $\beta = 0$, their combined effect occurs. Following the case of $\alpha = -1$ mentioned above, the calculation of the forward pass is perturbed, and its effect is propagated to the succeeding layers. In the backward pass, following the case of $\beta = 0$, the network parameters of only the selected layers are not updated. This can avoid *destructive* updates caused by negative α. Hence, $(-1, 0)$ is expected to be effective. Cases i and l contained $(-1, 0)$, and the former was the best.

By extending the discussion above, we can interpret the behavior of ShakeDrop using $\alpha = 0$ and $\beta \in [0, 1]$, which was the most effective on ResNet. When $\alpha = 0$, in the

TABLE 5. [Comparison on CIFAR datasets] Top-1 errors (%) on CIFAR datasets. This table shows the results of the original networks (left) and modified networks (right). Modified networks refer to the "EraseReLU"ed versions in (a) and (c) and networks in which BN was inserted at the end of residual branches in (b). In ShakeDrop, $\alpha = 0$, $\beta \in [0, 1]$ was used in the original networks and $\alpha \in [-1, 1]$, $\beta \in [0, 1]$ was used in the modified networks. In both cases, $p_L = 0.5$ and the pixel-level update rule were used. "×" indicates that learning did not converge. * indicates that the result is quoted from the literature. + indicates the average result of four runs.

(a) Two-branch architectures (ResNet, ResNeXt, and PyramidNet)

Methods	Regularization	CIFAR-10		CIFAR-100	
		Original	EraseReLU	Original	EraseReLU
ResNet-110 <Conv-BN-ReLU-Conv-BN-add-(ReLU)>	Vanilla	6.59	×	+27.42	+25.38
	RandomDrop	5.51	5.16	+24.07	+22.86
	ShakeDrop	**4.56**	**4.81**	+**23.74**	+**21.81**
ResNet-164 Bottleneck <Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)>	Vanilla	5.54	×	22.00	21.96
	RandomDrop	5.27	4.71	21.96	20.35
	ShakeDrop	**4.34**	**4.26**	**21.62**	**19.58**
ResNeXt-29 8-64d Bottleneck <Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)>	Vanilla	4.79	4.75	20.90	20.25
	RandomDrop	**4.38**	4.9	**20.66**	20.28
	ShakeDrop	**4.38**	**3.86**	20.71	**18.66**
PyramidNet-110 α270 <BN-Conv-BN-ReLU-Conv-BN-add>	Vanilla	3.85		+18.01	
	RandomDrop	3.63		+17.74	
	ShakeDrop	**3.33**		+**15.78**	
PyramidNet-272 α200 Bottleneck <BN-Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add>	Vanilla	3.53		*16.35	
	RandomDrop	3.41		15.94	
	ShakeDrop	**3.08**		**14.96**	

(b) Two-branch architectures (Wide-ResNet)

Methods	Regularization	CIFAR-10		CIFAR-100	
		Original	with BN	Original	with BN
Wide-ResNet-28-10k <BN-ReLU-Conv-BN-ReLU-Conv-(BN)-add>	Vanilla	**4.05**	3.98	20.67	20.05
	RandomDrop	4.15	**3.85**	**19.95**	**19.16**
	ShakeDrop	87.6	4.37	98.29	19.47

(c) Three-branch architectures

Methods	Regularization	CIFAR-10		CIFAR-100	
		Original	EraseReLU	Original	EraseReLU
ResNeXt-164 2-1-40d Bottleneck <Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)>	Vanilla	6.92	×	23.82	21.75
	RandomDrop Type-A	5.00	×	21.38	20.44
	RandomDrop Type-B	6.78	**4.58**	21.34	20.21
	Shake-Shake	5.61	4.65	22.35	22.51
	ShakeDrop Type-A	4.67	×	21.41	19.19
	ShakeDrop Type-B	**4.33**	×	21.52	**18.66**
ResNeXt-29 2-4-64d Bottleneck <Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)>	Vanilla	5.39	5.01	21.19	×
	RandomDrop Type-A	4.6	5.12	21.12	20.13
	RandomDrop Type-B	4.13	4.42	19.27	19.01
	Shake-Shake	4.64	3.84	19.16	18.82
	ShakeDrop Type-A	9.38	4.35	22.51	18.49
	ShakeDrop Type-B	**3.91**	**3.67**	**18.27**	**17.80**

forward pass, the outputs of the selected layers are identical to the inputs. In the backward pass, the amount of updating of the network parameters is perturbed by β.

V. EXPERIMENTS

A. COMPARISON ON CIFAR DATASETS

The proposed ShakeDrop was compared with RandomDrop and Shake-Shake in addition to the vanilla network (without regularization) on ResNet, Wide ResNet, ResNeXt, and Pyramidnet. Implementation details are available in Appendix A.

Table 5 shows the conditions and experimental results on CIFAR datasets [17]. In the table, method names are followed by the components of their building blocks. We used the parameters of ShakeDrop found in Section IV; that is, the original networks used $\alpha = 0, \beta \in [0, 1]$ and the modified networks in which the residual branches end with BN (e.g., EraseReLU versions) used $\alpha \in [-1, 1]$,

$\beta \in [0, 1]$. In ResNet and two-branch ResNeXt, in addition to the original form, EraseReLU versions were examined. In Wide ResNet, BN was added to the end of residual branches so that the residual branches ended with BN. In three-branch ResNeXt, we examined two approaches, referred to as "Type A" and "Type B," to apply RandomDrop and ShakeDrop. "Type A" and "Type B" indicate that the regularization unit was inserted after and before the addition unit for residual branches, respectively; that is, on the forward pass of the training phase, Type A is given by

$$G(x) = x + D(F_1(x) + F_2(x)), \qquad (10)$$

where $D(\cdot)$ is a perturbation unit of RandomDrop or Shake-Drop, and Type B is given by

$$G(x) = x + D_1(F_1(x)) + D_2(F_2(x)), \qquad (11)$$

where $D_1(\cdot)$ and $D_2(\cdot)$ are individual perturbation units.

TABLE 6. [Comparison on ImageNet] Top-1 errors (%) on ImageNet. This table shows the results of the original networks (left) and modified networks in which BN is at the end of the residual block (right). In ShakeDrop, $\alpha = 0$, $\beta \in [0, 1]$ were used in the original networks and $\alpha \in [-1, 1]$, $\beta \in [0, 1]$ in the modified networks. In both cases, $p_L = 0.9$ and the pixel-level update rule were used.

Methods	Regularization	Original	EraseReLU
ResNet-152 <Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)>	Vanilla	21.72	22.79
	RandomDrop	21.33	22.14
	ShakeDrop	**20.88**	**21.78**
ResNeXt-152 <Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)>	Vanilla	20.49	22.57
	RandomDrop	20.45	22.09
	ShakeDrop	**20.34**	**21.52**
PyramidNet-152 α300 <BN-Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add>	Vanilla	21.54	
	RandomDrop	21.23	
	ShakeDrop	**20.94**	

Table 5 shows that ShakeDrop can be applied not only to three-branch architectures (ResNeXt) but also two-branch architectures (ResNet, Wide ResNet, and PyramidNet), and ShakeDrop outperformed RandomDrop and Shake-Shake, except for some cases. In Wide ResNet with BN, although ShakeDrop improved the error rate compared with the vanilla network, it did not compared with RandomDrop. This is because the network only had 28 layers. As shown in the RandomDrop paper [14], RandomDrop is less effective on a shallow network and more effective on a deep network. We observed the same phenomenon in ShakeDrop, and Shake-Drop is more sensitive than RandomDrop. See Section V-E for more detail.

B. COMPARISON ON THE IMAGENET DATASET

We also conducted experiments on the ImageNet classification dataset [19] using ResNet, ResNeXt, and PyramidNet of 152 layers. The implementation details are presented in Appendix A. We used the best parameters found on the CIFAR datasets, except for p_L. We experimentally selected $p_L = 0.9$.

Table 6 shows the experimental results. Contrary to the CIFAR cases, the EraseReLU versions were worse than the original networks, which does not support the claim of the EraseReLU paper [3]. On ResNet and ResNeXt, in both the original and EraseReLU versions, ShakeDrop clearly outperformed RandomDrop and the vanilla network (ShakeDrop gained 0.84% and 0.15% compared with the vanilla network in the original networks, respectively). On PyramidNet, ShakeDrop outperformed the vanilla network (ShakeDrop gained 0.60% compared with the vanilla network) and also RandomDrop (ShakeDrop gained by 0.29% compared with RandomDrop). Therefore, on ResNet, ResNeXt, and Pyramid-Net, ShakeDrop clearly outperformed RandomDrop and the vanilla network.

C. COMPARISON ON THE COCO DATASET

From the results in Sections V-A and V-B, we considered that ShakeDrop promoted the generality of feature extraction and we evaluated the generality on the COCO dataset [22]. We used Faster R-CNN and Mask R-CNN with the

TABLE 7. [Comparison on COCO datasets] Average precision (%) on the COCO minival dataset. "Det." denotes the average precision of object detection and "Seg." denotes the average precision of instance segmentation.

Method	Regularization	Det.	Seg.
Faster R-CNN (with ResNet-152)	Vanilla	39.0	-
	RandomDrop	39.7	-
	ShakeDrop	**40.1**	-
Mask R-CNN (with ResNet-152)	Vanilla	39.7	35.8
	RandomDrop	40.8	36.6
	ShakeDrop	**40.9**	**36.9**

TABLE 8. [Mixup + ShakeDrop] Error rates (%) of mixup + ShakeDrop.

	Method	mixup	mixup + ShakeDrop
CIFAR-100	ResNet-110	**23.79**	24.12
	ResNet-164	22.93	**21.13**
	ResNeXt-29 8-64d	20.46	**17.52**
	PyramidNet-110 α270	17.47	**15.04**
	PyramidNet-272 α200	16.61	**14.90**
	ResNeXt-164 2-1-40d	22.13	**20.97** (Type B)
	ResNeXt-29 2-4-64d	20.54	**17.10** (Type B)
ImageNet	ResNet-152	21.46	**21.15**

ImageNet pre-trained original version ResNet of 152 layers in Section V-B. The implementation details are presented in Appendix A.

Table 7 shows the experimental results. On Faster R-CNN and Mask R-CNN, ShakeDrop clearly outperformed RandomDrop and the vanilla network. Therefore, ShakeDrop promoted the generality of feature extraction not only for image classification but also detection and instance segmentation.

D. SIMULTANEOUS USE OF SHAKEDROP WITH MIXUP

As mentioned in Section III-C, we have successfully used ShakeDrop combined with mixup. Table. 8 shows the results. In most cases, ShakeDrop further improved the error rates of the base neural networks to which mixup was applied. This indicates that ShakeDrop is not a rival to other regularization methods, such as mixup, but a "collaborator."

E. RELATIONSHIP BETWEEN NETWORK DEPTH AND BEST P_L

As mentioned in Section II, it has been experimentally found that RandomDrop is more effective on deeper networks (see

TABLE 9. [p_L and depth] Top-1 errors (%) of "ResNet + ShakeDrop" and "PyramidNet + ShakeDrop" at the final (300th) epoch. In ShakeDrop, $\alpha = 0$, $\beta \in [0, 1]$ were used in ResNet and $\alpha \in [-1, 1]$, $\beta \in [0, 1]$ in PyramidNet. CIFAR-100 dataset in the channel-level update rule. + indicates the average result of four runs.

Methods	Regularization	p_L	20 Layer	38 Layer	56 Layer	74 Layer	92 Layer	110 Layer
ResNet	Vanilla	-	31.76	29.03	28.39	27.36	26.28	+27.42
	RandomDrop	0.9	30.48	27.49	26.37	25.90	25.36	25.28
		0.8	**30.39**	**26.65**	26.14	24.92	25.50	24.54
		0.7	30.80	27.29	**25.18**	**24.50**	**24.70**	**23.80**
		0.6	32.61	26.97	25.88	24.67	24.77	24.39
		0.5	33.53	28.27	26.48	24.97	24.98	+24.07
	ShakeDrop	0.9	**29.66**	26.81	26.26	25.73	25.50	24.38
		0.8	30.97	26.93	26.00	25.26	25.09	24.28
		0.7	32.93	27.01	26.14	24.83	24.88	23.99
		0.6	33.47	27.40	**25.26**	24.73	24.54	24.21
		0.5	36.09	28.56	26.00	**24.57**	**23.98**	+**23.74**
PyramidNet	Vanilla	-	22.52	19.89	19.37	18.41	18.67	+18.01
	RandomDrop	0.9	21.00	**18.75**	18.71	17.59	17.70	**16.97**
		0.8	**20.88**	19.27	18.06	**17.49**	17.92	17.19
		0.7	21.08	19.02	18.47	17.89	**17.62**	17.20
		0.6	**20.88**	19.69	**17.89**	17.94	17.74	17.50
		0.5	21.88	19.87	18.52	18.06	17.69	+17.74
	ShakeDrop	0.9	20.40	18.39	17.70	17.42	17.62	17.15
		0.8	20.36	18.57	17.97	16.96	17.38	16.69
		0.7	20.96	**17.90**	17.08	16.69	16.56	16.34
		0.6	**20.29**	18.17	**17.07**	**16.50**	**16.42**	16.46
		0.5	20.54	17.93	17.35	16.73	16.47	+**15.78**

the figure on the right in Fig. 8 of [14]). We performed similar experiments on ShakeDrop and RandomDrop to compare their sensitivity to the depth of networks.

Table 9 shows that the error rates varied over both p_L and the network depth. ShakeDrop with a large p_L tended to be effective in shallower networks. The same observation was obtained in the experimental study on the relationship between p_L of RandomDrop and generalization performance [14]. We recommend a large p_L for shallower network architectures.

VI. CONCLUSION
We proposed a new stochastic regularization method called ShakeDrop which, in principle, can be applied to the ResNet family. Through experiments on the CIFAR and ImageNet datasets, we confirmed that, in most cases, ShakeDrop outperformed existing regularization methods of the same category, that is, Shake-Shake and RandomDrop.

APPENDIXES
APPENDIX A
EXPERIMENTAL CONDITIONS
All networks were trained using back-propagation by SGD with the Nesterov accelerated gradient [24] and momentum method [25]. Four GPUs (on CIFAR) and eight GPUs (on ImageNet) were used for learning acceleration: because of parallel processing, different observations of b_l, α, and β were obtained on each GPU. For example, the l-th layer on a GPU could be perturbed, whereas the layer was not perturbed on other GPUs (l is an arbitrary number). Additionally, even if the layer was perturbed on multiple GPUs, the differ-

ent observations of α and β could be used depending on each GPU.

All implementations used in the experiments were based on the publicly available code of ResNet,[3] ResNeXt,[4] PyramidNet,[5] Wide ResNet,[6] Shake-Shake,[7] and Faster/Mask R-CNN.[8] We changed their various learning conditions to make them as common as possible on CIFAR (in Section V-A). Table 10 shows the main changes. The implementation is available at https://github.com/imenurok/ShakeDrop.

The experimental conditions for each type of dataset are described below.

CIFAR datasets The input images of CIFAR datasets [17] were processed in the following manner. The original images of 32×32 pixels were color-normalized and then horizontally flipped with a 50% probability. Then, they were zero-padded to be 40×40 pixels and randomly cropped to be images of 32×32 pixels. On PyramidNet, the initial learning rate was set to 0.1 on CIFAR-10 and 0.5 on CIFAR-100 following the PyramidNet paper [8]. Other than PyramidNet, the initial learning rate was set to 0.1. The initial learning rate was decayed by a factor of 0.1 at 150 epochs and 225 epochs of the entire learning process (300 epochs), respectively. Additionally, a weight decay of 0.0001, momentum of 0.9, and batch size of 128 were used on four GPUs. "MSRA" [10]

[3] https://github.com/facebook/fb.resnet.torch
[4] https://github.com/facebookresearch/ResNeXt
[5] https://github.com/jhkim89/PyramidNet
[6] https://github.com/szagoruyko/wide-residual-networks
[7] https://github.com/xgastaldi/shake-shake
[8] https://github.com/facebookresearch/maskrcnn-benchmark

TABLE 10. [Learning conditions on CIFAR datasets] Learning conditions of the original experiments and our experiments. "Init." denotes the initial learning rates (CIFAR-10/CIFAR-100). "Total" denotes the total epoch number and "WD" denotes the weight decay. "it." denotes iterations and "ep." denotes epochs. "−" indicates that a value was not specified in the original paper. Bold items for our experiments indicate changes from the original conditions. We used the most common conditions for the original conditions, except for the initial learning rates and number of GPUs. Other than ResNet, the original learning rates were used in the experiments. On ResNet, the learning rate was 0.1 because our learning rate schedule did not warm up the training for the first 0.4k iterations. We used four GPUs to accelerate learning as much as possible.

Methods	Version	Init.	Learning rate schedule (operation timing)	Total	#GPU	WD
ResNet	Original [12]	0.01 / 0.01	$\times 10$ (0.4k it.)$\to \times 0.1$ (32k it.)$\to \times 0.1$ (48k it.)	-	2	0.0001
	Ours	**0.1 / 0.1**	**$\times 0.1$ (150 ep.)$\to \times 0.1$ (225 ep.)**	**300**	**4**	0.0001
ResNeXt	Original [33]	0.1 / 0.1	$\times 0.1$ (150 ep.)$\to \times 0.1$ (225 ep.)	300	8	0.0005
	Ours	0.1 / 0.1	$\times 0.1$ (150 ep.)$\to \times 0.1$ (225 ep.)	300	**4**	**0.0001**
PyramidNet	Original [8]	0.1 / 0.5	$\times 0.1$ (150 ep.)$\to \times 0.1$ (225 ep.)	300	-	0.0001
	Ours	0.1 / 0.5	$\times 0.1$ (150 ep.)$\to \times 0.1$ (225 ep.)	300	**4**	0.0001
Wide ResNet	Original [36]	0.1 / 0.1	$\times 0.2$ (60 ep.)$\to \times 0.2$ (120 ep.)$\to \times 0.2$ (160 ep.)	200	1	0.0005
	Ours	0.1 / 0.1	**$\times 0.1$ (150 ep.)$\to \times 0.1$ (225 ep.)**	**300**	**4**	**0.0001**

was used as the filter parameter initializer. We evaluated the top-1 errors without any ensemble technique. Linear decay parameter $p_L = 0.5$ was used following the RandomDrop paper [14]. ShakeDrop used parameters of $\alpha = 0$, $\beta = [0, 1]$ (Original) and $\alpha = [−1, 1]$, $\beta = [0, 1]$ (EraseReLU on ResNet and ResNeXt, Wide ResNet with BN, and Pyramid-Net) with the pixel-level update rule.

ImageNet dataset The input images of ImageNet [1] were processed in the following manner. The original image was distorted using a random aspect ratio [27] and randomly cropped to an image size of 224×224 pixels. Then, the image was horizontally flipped with a 50% probability and standard color noise [19] was added. On PyramidNet, the initial learning rate was set to 0.5. The initial learning rate was decayed by a factor of 0.1 at 60, 90, and 105 epochs of the entire learning process (120 epochs) following [8]. Additionally, a batch size of 128 was used on eight GPUs. Other than PyramidNet, the initial learning rate was set to 0.1. The initial learning rate was decayed by a factor of 0.1 at 30, 60, and 80 epochs of the entire learning process (90 epochs) following [7]. Additionally, a batch size of 256 was used on eight GPUs. A weight decay of 0.0001 and momentum of 0.9 were used. "MSRA" [10] was used as the filter parameter initializer. We evaluated the top-1 errors without any ensemble technique on the single 224×224 image that was cropped from the center of an image resized with the shorter side 256. $p_L = 0.9$ was used as the linear decay parameter. ShakeDrop used parameters of $\alpha = 0$, $\beta = [0, 1]$ (Original) and $\alpha = [−1, 1]$, $\beta = [0, 1]$ (EraseReLU on ResNet and ResNeXt, and PyramidNet) with the pixel-level update rule.

COCO dataset Input images of COCO [22] were processed in the following manner. We trained models on the union of the 80k training set and 35k val subset, and evaluated the models on the remaining 5k val subset. We used ResNet-152 for the backbone network and FPN [21] for the predictor network. To use ResNet-152 as a feature extractor, we used the expected value $E(b_l + \alpha − b_l\alpha)$ instead of ShakeDrop regularization. According to the experimental condition of the ImageNet dataset, the original image was color-normalized with the means and standard deviations of

ImageNet dataset images. The initial learning rate was set to 0.2. The initial learning rate was decayed by a factor of 0.1 at 60, 000 and 80, 000 iterations of the entire learning process (90,000 iterations). Additionally, a batch size of 16 was used on eight GPUs. A weight decay of 0.0001 was used. The other experimental conditions were set according to *maskrcnn-benchmark*[8].

REFERENCES
[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009.
[2] T. DeVries and G. W. Taylor, "Dataset augmentation in feature space," in *Proc. ICLR Workshop*, 2017.
[3] X. Dong, G. Kang, K. Zhan, and Y. Yang, "EraseReLU: A simple way to ease the training of deep convolution neural networks," 2017, *arXiv:1709.07634*. [Online]. Available: https://arxiv.org/abs/1709.07634
[4] X. Gastaldi, "Shake-shake regularization," 2017, *arXiv:1705.07485v2*. [Online]. Available: https://arxiv.org/abs/1705.07485v2
[5] X. Gastaldi, "Shake-shake regularization of 3-branch residual networks," in *Proc. ICLR workshop*, 2017.
[6] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. ICLR*, 2015.
[7] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training imagenet in 1 hour," 2017, *arXiv:1706.02677*. [Online]. Available: https://arxiv.org/abs/1706.02677
[8] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," in *Proc. CVPR*, 2017.
[9] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," 2017, *arXiv:1610.02915v4*. [Online]. Available: https://arxiv.org/abs/1610.02915v4
[10] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. ICCV*, 2015.
[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.
[12] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. ECCV*, pp. 630–645, 2016.
[13] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep networks with stochastic depth," 2016, *arXiv:1603.09382v3*. [Online]. Available: https://arxiv.org/abs/1603.09382v3
[14] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. ECCV*, 2016.
[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, 2015.

[16] R. Kleinberg, Y. Li, and Y. Yuan, "An alternative view: When does SGD escape local minima?" in *Proc. ICML*, 2018.

[17] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012.

[20] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Proc. NIPS*, 1992.

[21] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. CVPR*, 2017.

[22] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. P. Dollár, "Microsoft COCO: Common objects in context," in *Proc. ECCV*, 2014.

[23] T. Miyato, S.-I. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: A regularization method for supervised and semi-supervised learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 8, pp. 1979–1993, Aug. 2019.

[24] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$," *Soviet Math. Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

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

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

[29] R. Takahashi, T. Matsubara, and K. Uehara, "RICAP: Random image cropping and patching data augmentation for deep CNNs," in *Proc. 10th Asian Conf. Mach. Learn., Mach. Learn. Res.*, vol. 95, J. Zhu and I. Takeuchi, Eds., Nov. 2018, pp. 786–798.

[30] Y. Tokozume, Y. Ushiku, and T. Harada, "Between-class learning for image classification," in *Proc. CVPR*, 2018.

[31] V. Verma, A. Lamb, C. Beckham, A. Courville, I. Mitliagkis, and Y. Bengio, "Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer," 2018, *arXiv:1806.05236*. [Online]. Available: https://arxiv.org/abs/1806.05236

[32] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using DropConnect," in *Proc. ICML*, 2013.

[33] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. CVPR*, 2017.

[34] Y. Yamada, M. Iwamura, and K. Kise, "Deep pyramidal residual networks with separated stochastic depth," 2016, *arXiv:1612.01230*. [Online]. Available: https://arxiv.org/abs/1612.01230

[35] Y. Yamada, M. Iwamura, and K. Kise, "Shakedrop regularization," in *Proc. ICLR Workshop*, 2018.

[36] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. BMVC*, 2016.

[37] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *Proc. ICLR*, 2017.

[38] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *Proc. ICLR*, 2018.

MASAKAZU IWAMURA (M'04) received the B.E., M.E., and Ph.D. degrees in engineering from Tohoku University, Japan, in 1998, 2000, and 2003, respectively.

He is currently an Associate Professor with the Department of Computer Science and Intelligent Systems, Graduate School of Engineering, Osaka Prefecture University.

Dr. Iwamura is also a member of IPSJ and IEICE. He was a recipient of the Best Paper Award of the IEICE, in 2006, the IAPR/ICDAR Best Paper Awards, in 2007, the IAPR Nakano Award (Best Paper Award), in 2010, the ICFHR Best Paper Award, in 2010, the IAPR/ICDAR Young Investigator Award, in 2011, the MVA Best Paper Award, in 2017, and the MIRU Excellence Student Award, in 2018. He has contributed to the community in various roles; he has served as an Organizer of conferences/workshops by serving as a Co-Chair of the IWRR 2014 and 2016, and the CBDAR 2011 and 2013. He has served the Track Chair of the ICPR 2012 and the ICDAR 2015, an Executive Chair and the Financial Chair of the ICDAR 2017, a Program Co-Chair of the CJKPR 2010, and an Organizer of the 2013, 2015, and 2017 Robust Reading Competitions. He has been an Associate Editor of the *International Journal of Document Analysis and Recognition*, since 2013. He has been a Guest Editor for a special issue on Robust Reading of the journal, in 2015, and an Associate Editor of the IEICE *Transactions on Information and Systems*, since 2017. He was the Webserver, from 2009 to 2016, and the Vice-Chair of IAPR TC-11 (Reading Systems), from 2016 to 2018, while serving as a Regular Member of Program Committees of ICDAR, DAS, ICFHR, and CBDAR.
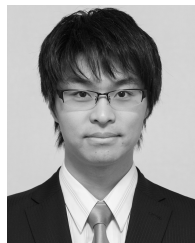
TAKUYA AKIBA received the B.S. degree in information science and the M.S. and Ph.D. degrees in computer science from The University of Tokyo, Japan, in 2011, 2013, and 2015, respectively.

From 2015 to 2016, he was an Assistant Professor with the National Institute of Informatics, Japan. Since 2016, he has been a Researcher with Preferred Networks, Inc., Japan. His research interest includes research and development of machine learning systems.

Dr. Akiba received the awards and honors, including Dean's Award from the University of Tokyo, in 2015, the Best Paper Award from the 6th Forum on Data Engineering and Information Management, in 2014, and the JSPS Ikushi Prize, in 2015.

KOICHI KISE (M'02) received the B.E., M.E., and Ph.D. degrees in communication engineering from Osaka University, Osaka, Japan, in 1986, 1988, and 1991, respectively.

From 2000 to 2001, he was a Visiting Professor with the German Research Center for Artificial Intelligence (DFKI), Germany. He is currently a Professor with the Department of Computer Science and Intelligent Systems and the Director of the Institute of Document Analysis and Knowledge Science (IDAKS), Osaka Prefecture University, Japan. His major research activities are in analysis, recognition and retrieval of documents, images and activities.

Prof. Kise is also a member of IEICE, ACM, IPSJ, IEEJ, ANLP, and HIS. He was a recipient of the Best Paper Award of IEICE, in 2006, the IAPR/ICDAR Best Paper Awards, in 2007 and 2013, the IAPR Nakano Award, in 2010, the ICFHR Best Paper Award, in 2010, and the ACPR Best Paper Award, in 2011. He worked as the Chair of the IAPR Technical Committee 11 (reading systems), and a member of the IAPR conferences and meetings committee. He is currently the Editor-In-Chief of the international journal of document analysis and recognition.

YOSHIHIRO YAMADA received the B.S. and M.S. degrees in engineering from Osaka Prefecture University, Osaka, Japan, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree in engineering.

His research interests include the development of digital image analysis techniques for image classification and fundamental study of neural networks.

Mr. Yamada was a recipient of the PRMU Research Encouragement Award, in 2016 and 2017, the MIRU Interactive Presentation Award, in 2017, and the MIRU Excellence Student Award, in 2018.

● ● ●