ShakeDrop Regularization for Deep Residual Learning

Yoshihiro Yamada\textsuperscript{1} Masakazu Iwamura\textsuperscript{1} Takuya Akiba\textsuperscript{2} Koichi Kise\textsuperscript{1}
\textsuperscript{1}Osaka Prefecture University \textsuperscript{2}Preferred Networks, Inc.

Abstract

Overfitting is a crucial problem in deep neural networks, even in the latest network architectures. In this paper, so as to relieve the overfitting effect of ResNet and its improvements (i.e., Wide ResNet, PyramidNet and ResNeXt), we propose a new regularization method, named ShakeDrop regularization. ShakeDrop is inspired by Shake-Shake, which is an effective regularization method but can be applied to only ResNeXt. ShakeDrop is even more effective than Shake-Shake and can be successfully applied to not only ResNeXt but also ResNet, Wide ResNet and PyramidNet. The important key to realize ShakeDrop is stability of training. Since effective regularization often causes unstable training, we introduce a stabilizer of training which is an unusual usage of an existing regularizer. Experiments reveals that ShakeDrop achieves comparable or superior generalization performance to conventional methods\textsuperscript{1}.

1. Introduction

Recent advances in generic object recognition have been brought by deep neural networks. After ResNet\textsuperscript{[11]} opened the door to very deep CNNs of over a hundred layers by introducing the building block, its improvements such as Wide ResNet\textsuperscript{[29]}, PyramidNet\textsuperscript{[8, 9]} and ResNeXt\textsuperscript{[27]} have broken the records of lowest error rates. The development of such base network architectures, however, is not enough to reduce the generalization error (i.e., difference between the training and test error) sufficiently. It is thought that some regularization methods are required to reduce the generalization error\textsuperscript{[30]}. Widely-used regularization methods include data augmentation\textsuperscript{[16]}, stochastic gradient descent (SGD)\textsuperscript{[30]}, weight decay\textsuperscript{[18]}, batch normalization (BN)\textsuperscript{[14]}, label smoothing\textsuperscript{[23]}, adversarial training\textsuperscript{[6]}, mixup\textsuperscript{[31, 24, 25]} and dropout\textsuperscript{[21, 26]}.\footnote{ShakeDrop achieved the state-of-the-art performance in the CIFAR-10/100 datasets. Please refer to Version 1 on arXiv: https://arxiv.org/abs/1802.02375}

Recently, an effective regularization method named Shake-Shake regularization\textsuperscript{[5, 4]} was proposed. It is an interesting method which, in the training, disturbs calculation of the forward pass by using a random variable, and also that of backward pass by a different random variable. Its effectiveness is proven by the experiment that ResNeXt to which Shake-Shake is applied (hereafter, this is denoted by “ResNeXt + Shake-Shake”) achieved the lowest error rate on CIFAR-10/100 datasets\textsuperscript{[15]}. Shake-Shake, however, has following two drawbacks; (1) it can be applied only to ResNeXt, and (2) the reason it is effective is not yet revealed.

The current paper addresses these problems. For the problem (1), we propose a novel powerful regularization method \textit{ShakeDrop regularization}, which is more effective than Shake-Shake. Its big advantage is that it can be applied not only to ResNeXt (hereafter, 3-branch architectures) but also ResNet, Wide ResNet and PyramidNet (2-branch architectures). For the problem (2), in the process of deriving ShakeDrop, we give an intuitive interpretation of Shake-Shake. Through experiments using various base network architectures, we reveal the condition that the proposed ShakeDrop successfully works.

2. Regularization methods for ResNet family

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

\textbf{Shake-Shake regularization}\textsuperscript{[5, 4]} is an effective regularization method for ResNeXt. It is illustrated in Fig. 1(a). The basic ResNeXt building block which has a 3-branch architecture is given as

\begin{align}
G(x) = x + F_1(x) + F_2(x),
\end{align}

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

Letting \(\alpha\) and \(\beta\) be independent random coefficients uniformly drawn in the range of \([0, 1]\), Shake-Shake is given

\begin{align}
\alpha x + (1 - \alpha)x + \beta F_1(x) + (1 - \beta)F_2(x),
\end{align}

where \(G(x)\) is the output of the building block. This notation is applied to all combinations of base network architectures and regularization methods.
Figure 1. Regularization methods for ResNet family. Conv represents convolution layer, \( E[x] \) expected value of \( x \), and \( \alpha \), \( \beta \) and \( b_l \) random coefficients which are changed on every iteration. (a) and (b) are existing methods. (c) is an intermediate regularization method \textit{Single-branch Shake} to the proposed method. (d) is the proposed method.

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

where \textit{train-fwd} and \textit{train-bwd} mean the forward and backward passes of training, respectively. The expected values \( E[\alpha] = E[1 - \alpha] = 0.5 \). Eqn. (2) means that calculation of the forward pass is multiplied by a random coefficient \( \alpha \) and that of the backward pass by another random coefficient \( \beta \). The values of \( \alpha \) and \( \beta \) are drawn in every image or batch. The paper suggests to train 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 a coefficient \( \alpha \) in the forward pass, it is natural to multiply the gradient by the same coefficient in the backward pass. Hence, Shake-Shake regularization makes the gradient \( \beta / \alpha \) 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 from the network parameters being captured in local minima. However, the reason why such disturbance is effective is not sufficiently revealed.

**RandomDrop regularization (a.k.a., Stochastic Depth and ResDrop)** [13] is a regularization method originally proposed for ResNet, and also applied to PyramidNet [28]. It is illustrated in Fig. 1(b). The basic ResNet building block which has a 2-branch architecture is given as

\[
G(x) = x + F(x) 
\]

where \( F(x) \) is the output of the residual branch. RandomDrop makes the network apparently shallow in learning by dropping some building blocks stochastically selected. The \( l \)th building block from the input layer is given as

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

where \( b_l \in \{0, 1\} \) is a Bernoulli random variable with the probability \( P(b_l = 1) = E[b_l] = p_l \).

The paper recommends \textit{linear decay rule} to determine \( p_l \), which is given as

\[
p_l = 1 - \frac{l}{L} (1 - p_L),
\]

where \( L \) is the number of all building blocks, and \( p_L \) is the initial parameter. The paper suggests to use \( p_L = 0.5 \).

RandomDrop can be regarded as a simplified version of dropout [21]. These main difference is that while RandomDrop drops layers, dropout does elements.
3. Proposed Method

3.1. ShakeDrop Regularization

The proposed ShakeDrop, illustrated in Fig. 1(d), is given as

\[
G(x) = \begin{cases} 
  x + (b_1 + \alpha - b_1\alpha)F(x), & \text{in train-fwd} \\
  x + (b_1 + \beta - b_1\beta)F(x), & \text{in train-bwd} \\
  x + E[b_1 + \alpha - b_1\alpha]F(x), & \text{in test,}
\end{cases}
\]

where \( b_1 \) is a Bernoulli random variable with the probability \( P(b_1 = 1) = E[b_1] = p_1 \) given by the linear decay rule (Eqn. 5), and \( \alpha \) and \( \beta \) are independent uniform random variables. The most representative ranges of \( \alpha \) and \( \beta \) are (1) \( \alpha = 0, \beta \in [0, 1] \) and (2) \( \alpha \in [-1, 1], \beta \in [0, 1] \).

Further detail about parameters is presented in Sec. 4.

In the training phase, \( b_1 \) controls the behaviour of ShakeDrop. If \( b_1 = 1 \), Eqn. (6) is deformed as

\[
G(x) = \begin{cases} 
  x + F(x), & \text{in train-fwd} \\
  x + F(x), & \text{in train-bwd.}
\end{cases}
\]

That is, ShakeDrop is equivalent to the original network (e.g., ResNet). If \( b_1 = 0 \), Eqn. (6) is deformed as

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

That is, calculation of \( F(x) \) is perturbated by \( \alpha \) and \( \beta \).

In the following sections, we present how ShakeDrop is derived in Sec. 3.2 and discuss the relationship with existing regularization methods in Sec. 3.3.

3.2. Derivation of ShakeDrop

3.2.1 Interpretation of Shake-Shake regularization

We give an intuitive interpretation of the forward pass of Shake-Shake. To the best of our knowledge, it has not been given yet. As shown in Eqn. (2) (and also in Fig. 1(a)), in the forward pass, Shake-Shake interpolates the outputs of two residual branches (i.e., \( F_1(x) \) and \( F_2(x) \)) with a random weight \( \alpha \). As DeVries & Taylor [2] demonstrated that interpolation of two data in the feature space can synthesize reasonable augmented data, the interpolation of two residual branches of Shake-Shake in the forward pass can be interpreted as synthesizing data. Use of a random weight \( \alpha \) enables us to generate many different augmented data. On the other hand, in the backward pass, a different random weight \( \beta \) is used to disturb learning to make the network learnable long time.

3.2.2 Single-branch Shake Regularization

The regularization mechanism of Shake-Shake relies on two or more residual branches, so that it can be applied only to 2-branch networks architectures (i.e., ResNeXt). In order to realize a similar regularization to Shake-Shake on 2-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. Actually, DeVries & Taylor [2] demonstrated that not only interpolation but also noise addition in the feature space generates reasonable augmented data. Hence, following Shake-Shake, we apply random perturbation to the output of a residual branch (i.e., \( F(x) \) of Eqn. (3)). That is, it is given as

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

We call this regularization method Single-branch Shake. Single-branch Shake is expected to be effective like 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, the result on the CIFAR-100 dataset was hopelessly bad (i.e., an error rate of 77.99%).

3.2.3 Stabilization of training

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. It must be “two residual branches.” Let us argue if it makes sense. As presented in Sec. 2, in the training, Shake-Shake makes the gradients of two branches \( \beta/\alpha \) times and \( (1 - \beta)/(1 - \alpha) \) times as large as the correctly calculated gradients. Thus, when \( \alpha \) is close to 0 or 1, it can spoil training because it can make a gradient prohibitively large\(^3\). However, two residual branches of Shake-Shake work as a fail-safe system. That is, even if the coefficient on one branch (let us assume \( \beta/\alpha \)) is large, the other (i.e., \( (1 - \beta)/(1 - \alpha) \)) is kept small. In other words, training on at least one branch is not spoiled. However, Single-branch Shake does not have such fail-safe system.

From the discussion above, failure of Single-branch Shake is caused by too strong perturbation and lack of stabilizing mechanism. However, weakening the perturbation would also weaken the effect of regularization. Thus, we need a trick to stabilize learning under strong perturbation.

\(^3\)This idea is supported by an experiment conducted in the Shake-Shake paper [4]. They limited the ranges of \( \alpha \) and \( \beta \). When \( \alpha \) and \( \beta \) are kept close (more precisely, on the number line, \( \alpha \) and \( \beta \) are on the same side of 0.5 such as \( \alpha = 0.1 \) and \( \beta = 0.2 \)), Shake-Shake achieved relatively high accuracy. However, when \( \alpha \) and \( \beta \) are kept far (\( \alpha \) and \( \beta \) are on the other side of 0.5 such as \( \alpha = 0.1 \) and \( \beta = 0.7 \)), it did relatively low accuracy. This indicates that when \( \beta/\alpha \) or \( (1 - \beta)/(1 - \alpha) \) is large, training can become less stable.
Table 1. Regularization methods generating new data. “Sample-wise generation” means that data is generated using single sample.

| Regularization method | Data augmentation | Sample-wise generation |
|-----------------------|-------------------|-----------------------|
|                       | In (input) data space | In feature space | In label space |
| Data augmentation [16] | ✓ | ✓ | ✓ |
| Adversarial training [6] | ✓ | ✓ | |
| Label smoothing [23] | ✓ | ✓ | ✓ |
| Mixup [31, 24] | ✓ | ✓ | ✓ |
| Manifold mixup [25] | ✓ | ✓ | ✓ |
| Shake-Shake [5, 4] | ✓ | ✓ | ✓ |
| ShakeDrop | ✓ | ✓ | ✓ |

Our idea is to use the mechanism of RandomDrop for solving the issue. In our situation, however, the original usage of RandomDrop\(^4\) does not contribute because a shallower version of a strongly perturbated 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 of following two network architectures.

1. The original network (e.g., PyramidNet), which corresponds to Eqn. (7).
2. A network suffering from strong perturbation (e.g., “PyramidNet + Single-branch Shake”), which corresponds to Eqn. (8).

By mixing them up, it is expected that (1) when the original network is selected, learning is correctly promoted, (2) when the network with strong perturbation is selected, learning is disturbed. To achieve a good performance, two networks should be well balanced, which is controlled by a parameter \(p_L\). We will argue this issue in Secs. 4 and 5.

3.3. Relationship with existing regularization methods

In this section, we argue the relationship between ShakeDrop and existing regularization methods. Among them, SGD and weight decay are commonly used techniques in the training of deep neural networks. Though they are not designed for regularization, it is pointed out that they have generalization effects [30, 18]. BN [14] is a strong regularization technique which is widely used in recent network architectures. The proposed ShakeDrop will be appended to these regularization methods.

ShakeDrop differs from RandomDrop [13] and dropout [21, 26] in following two points. One is that they do not explicitly generate new data. The other is that they do not update network parameters based on noisy gradients.

Some methods regularize by generating new data. They are summarized in Table 1. Data augmentation [16] and adversarial training [6] synthesize data in the (input) data space. They differ in how they generate data. While the former uses manually designed means such as random crop and horizontal flip, the latter automatically generates data that should be used for training to improve the generalization performance. Label smoothing [23] generates (changes) labels for existing data. The methods mentioned above generate new data using single sample. On the other hand, some methods requires multiple samples to generate new data. Mixup and BC learning [31, 24] generate new data and their corresponding class labels by interpolating two data. Though they generate new data in the data space, manifold mixup [25] does it also in the feature space. Compared with ShakeDrop which generates data in the feature space using single sample, none of these regularization methods is in the same category except Shake-Shake.

4. Preliminary Experiments

The proposed ShapeDrop regularization has three parameters: \(\alpha\), \(\beta\) and \(p_L\). In addition, there are four possible update rules of \(\alpha\) and \(\beta\). In this section, we search the best parameters except \(p_L\) and best update rule on CIFAR-100 dataset. Following RandomDrop regularization [13], we used \(p_L = 0.5\) as default.

4.1. Ranges of \(\alpha\) and \(\beta\)

In this section, the best parameter ranges of \(\alpha\) and \(\beta\) are experimentally explored. We applied ShakeDrop to three network architectures: ResNet, ResNet (EraseReLU version) and PyramidNet. EraseReLU erases Rectified Linear Unit (ReLU) in the bottom of the building blocks to improve the generalization performance [3]. Note that EraseReLU does not affect PyramidNet because PyramidNet does not have ReLU in the bottom of the building blocks.

Table 2 shows representative parameter ranges of \(\alpha\) and \(\beta\) we tested and their results. Cases A and B correspond to the vanilla network (i.e., without regularization) and RandomDrop, respectively. On all three network architectures, case B was better than case A. Let us see the results of three network architectures one by one.

\(^4\)RandomDrop is designed to make a network apparently shallow so as to avoid problems of vanishing gradients, diminishing feature reuse and long training time.
Table 2. [Ranges of \( \alpha \) and \( \beta \)] Average Top-1 errors (%) of “ResNet + ShakeDrop,” “ResNet (EraseReLU version) + ShakeDrop,” and “PyramidNet + ShakeDrop” of 4 runs at the final (300th) epoch on CIFAR-100 dataset in the batch-level update rule. “×” means learning did not converge. Cases A and B are equivalent to without regularization method (we call this Vanilla) and RandomDrop, respectively.

| Case | \( \alpha \) | \( \beta \) | 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      |       |

- **PyramidNet** achieved the lowest error rates among 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. That is, case O which was the best on PyramidNet resulted in the chance rate (99.00%). 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, and case G was the best.

Through experiments using various base network architectures shown in Sec. 5, we found that case O is effective on “EraseReLU”ed architectures. On the other hand, case G is effective on non-“EraseReLU”ed architectures.

### 4.2. Update rule of \( \alpha \) and \( \beta \)

The best update rule of \( \alpha \) and \( \beta \) are found from batch, image, channel and pixel levels. They mean that the same \( \alpha \) and \( \beta \) are used for “all the images in the mini-batch,” “each image,” “each channel,” “each element” for each building block, respectively.

In the experiment, the best \( \alpha \) and \( \beta \) found in Sec. 4.1 (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\(^5\). Table 3 shows that the pixel level was the best on both ResNet and PyramidNet.

Table 3. [Update rule of \( \alpha \) and \( \beta \)] Average Top-1 errors (%) of “PyramidNet + ShakeDrop” of 4 runs at the final (300th) epoch on CIFAR-100 dataset.

| \( \alpha \) | \( \beta \) | Level | ResNet | PyramidNet |
|------------|------------|-------|--------|------------|
| 0          | [0, 1]     | Batch | 23.77  | -          |
|            |            | Image | 99.00  | -          |
|            |            | Channel | 66.30  | -          |
|            |            | Pixel | 23.74  | -          |

Then, we basically conduct the following experiments using \( \alpha \in [-1, 1] \), \( \beta \in [0, 1] \) and the pixel-level update rule.

### 4.3. Combinations of \((\alpha, \beta)\) for analyzing ShakeDrop behaviour

Though we have successfully found effective ranges of \( \alpha \) and \( \beta \) and their update rule, we still do not grasp what mechanism contributes to improve the generalization performance on ShakeDrop. One reason is that \( \alpha \) and \( \beta \) are random variables. Due to this, what we can obtain in the end of training is a network which is trained using various observed values of \( \alpha \) and \( \beta \). It makes more difficult to know the mechanism. Hence, in this section, we explored effective combinations of \((\alpha, \beta)\).

The combinations of \((\alpha, \beta)\) are defined as follows. From the best ranges of \( \alpha \) and \( \beta \) for PyramidNet which are \( \alpha \in [-1, 1] \) and \( \beta \in [0, 1] \), by taking both ends of ranges, we ob-

\(^5\) We did not conduct experiments in the following setting because they resulted in the chance rate in the experiments in Sec. 4.1; \( \alpha = 0 \) and \( \beta \in [0, 1] \) for PyramidNet, and \( \alpha \in [-1, 1] \) and \( \beta \in [0, 1] \) for ResNet. In Table 3, they are denoted by a hyphen (“-”).
tain a set of \((\alpha, \beta)\) pairs: \{\((1, 1), (1, 0), (-1, 1), (-1, 0)\}\}. Then, we examine its all combinations, which are shown in Table 4. Intuitively, instead of drawing \(\alpha\) and \(\beta\) in the ranges, the values of \(\alpha\) and \(\beta\) are picked up from the pool of \((\alpha, \beta)\) with equal probability.

Table 4 shows combinations of \((\alpha, \beta)\) and their results. First of all, compared with the best result in Table 2 (i.e., case O; 16.22%), the results in Table 4 were almost comparable. In particular, the best result in Table 4 (i.e., case i; 16.24%) was almost equivalent.

Then, we focus on cases i, j and k. They all have two marked elements and one of them is \((1, 1)\). Actually, \((1, 1)\) is the normal state of neural networks that is given by Eqn. (6). Hence, considering their difference means considering difference among \((-1, 0), (-1, 1)\) and \((1, 0)\).

**What does \((\alpha, \beta) = (1, 0)\) do?**
Let us consider the case of \(\beta = 0\). In such a case, the network parameters of the layers selected to perturbate (i.e., the layers with \(b_i = 0\)) are not updated. However, in the forward pass, their succeeding layers are also perturbated, and in the backward pass, their network parameters are updated reflecting the perturbation.

**What does \((\alpha, \beta) = (-1, 1)\) do?**
Let us consider the effect of \(\alpha = -1\). In the perturbated layers, calculation of the forward pass is perturbated by \(\alpha = -1\). Then, the effect of perturbation is propagated to succeeding layers. Hence, not only the perturbated layers but also their succeeding layers are perturbated. In the backward pass, when \(\alpha\) is negative, the network parameters of the perturbated layers can be updated toward the opposite direction to the normal one. Due to this, the network parameters of the perturbated layers are strongly affected by negative \(\alpha\).

**What does \((\alpha, \beta) = (-1, 0)\) do?**
A combined effect of \((1, 0)\) and \((-1, 1)\) would happen. First, calculation of the forward pass is perturbated. Its effect is propagated to succeeding layers. However, in the backward pass, only the network parameters of other than the perturbated layers are updated. Those of the perturbated layers are not updated. This can avoid dangerous updates by a large ratio mentioned in Sec. 3.2.3 because \(\beta / \alpha\) becomes 0.

Consequently, considering the effects of \((-1, 0), (-1, 1)\) and \((1, 0)\), \((-1, 0)\) works the best.

In addition, we observed that \(p_L\) matters to error rates. As mentioned above, \((1, 1)\) is the normal state. Hence, the group of cases i, j and k and the one of cases l, m and n only differ in \(p_L\); since the former group have two elements and the latter group do one element, \(p_L\) of the former group works as \(p_L / 2\). Comparing error rates between two groups, it shows that \(p_L\) greatly affects the error rates.

### 5. Experiments

#### 5.1. Comparison on CIFAR-100

The proposed ShakeDrop is compared with the vanilla network (without regularization), RandomDrop and ShakeShake in different network architectures including ResNet, Wide ResNet, ResNeXt and PyramidNet.

Table 5 shows the experimental results on CIFAR-100.
Table 5. Top-1 errors (%) on CIFAR-100 dataset. This table shows the results of the original networks (left) and modified networks (right). The modified networks mean the “EraseReLU”ed versions in (a) and (c) and the ones in which BN was inserted at the end of residual branches in (b). 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, $p_L = 0.5$ and the pixel-level update rule was used. “×” means learning did not converge. * indicates the result is quoted from the literature. + indicates the average result of 4 runs.

(a) 2-branch architectures (ResNet, ResNeXt and PyramidNet)

| Methods                        | Regularization | Original    | EraseReLU   |
|-------------------------------|----------------|-------------|-------------|
| **ResNet-110**                | Vanilla        | 27.42       | 25.38       |
| $<$Conv-BN-ReLU-Conv-BN-add-(ReLU)$> | ResDrop        | 24.07       | 22.86       |
|                               | ShakeDrop      | 23.74       | 21.81       |
| **ResNet-164 Bottleneck**     | Vanilla        | 22.00       | 21.96       |
| $<$Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)$> | ResDrop        | 21.96       | 20.35       |
|                               | ShakeDrop      | 21.62       | 19.58       |
| **ResNeXt-29 8-64d Bottleneck** | Vanilla        | 20.90       | 20.25       |
| $<$Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)$> | ResDrop        | 20.66       | 20.28       |
|                               | ShakeDrop      | 20.71       | 18.66       |
| **PyramidNet-110 α200 Bottleneck** | Vanilla        | *18.01      |             |
| $<$BN-Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add$> | ResDrop        | *17.74      |             |
|                               | ShakeDrop      | *15.78      |             |
| **PyramidNet-272 α200 Bottleneck** | Vanilla        | 16.35       |             |
| $<$BN-Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add$> | ResDrop        | 15.94       |             |
|                               | ShakeDrop      | 14.96       |             |

(b) 2-branch architectures (Wide-ResNet)

| Methods                        | Regularization | Original    | with BN     |
|-------------------------------|----------------|-------------|-------------|
| **Wide-ResNet-28-10k**        | Vanilla        | 26.49       | 24.24       |
| $<$BN-ReLU-Conv-BN-ReLU-Conv-(BN)-add$> | ResDrop        | 34.19       | 26.64       |
|                               | ShakeDrop      | 95.44       | 20.50       |

(c) 3-branch architectures

| Methods                        | Regularization | Original    | EraseReLU   |
|-------------------------------|----------------|-------------|-------------|
| **ResNeXt-164 2-1-40d Bottleneck** | Vanilla        | 23.82       | 21.75       |
| $<$Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)$> | ResDrop Type-A | 21.38       | 20.44       |
|                               | ResDrop Type-B | 21.34       | 20.21       |
|                               | Shake-Shake    | 22.35       | 22.51       |
|                               | ShakeDrop Type-A | 21.41    | 19.19       |
|                               | ShakeDrop Type-B | 21.52    | 18.66       |
| **ResNeXt-29 2-4-64d Bottleneck** | Vanilla        | 21.19       | ×           |
| $<$Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)$> | ResDrop Type-A | 21.12       | 20.13       |
|                               | ResDrop Type-B | 19.27       | 19.01       |
|                               | Shake-Shake    | 19.16       | 18.82       |
|                               | ShakeDrop Type-A | 22.51    | 18.49       |
|                               | ShakeDrop Type-B | 18.27    | 17.80       |
Table 6. Top-1 errors (%) on ImageNet at 90th epoch. This table shows the results of the original networks (left) and modified networks in which BN is at the end of 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, $p_L = 0.5$ and the pixel-level update rule was used.

| Methods                  | Regularization | $p_L$ | Original | EraseReLU |
|--------------------------|----------------|-------|----------|-----------|
| **ResNet-50**            |                |       |          |           |
| <Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)> | Vanilla        | -     | 23.60    | 25.63     |
|                          | RandomDrop     | 0.05  | 23.36    | 25.61     |
|                          | ShakeDrop      | 0.025 | 23.39    | 25.66     |
|                          |                | 0.025 | 23.33    | 25.69     |
|                          |                |       | **23.13**| **25.46** |
| **ResNetXt-50 32-4d**    |                |       |          |           |
| <Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add-(ReLU)> | Vanilla        | -     | 22.78    | 24.66     |
|                          | RandomDrop     | 0.05  | 22.47    | 24.45     |
|                          | ShakeDrop      | 0.025 | 22.55    | 24.50     |
|                          |                |       | **22.38**| **24.41** |
|                          |                |       | **22.19**| **24.27** |
| **PyramidNet-50 $\alpha$300** | Vanilla | -     | 24.06    |           |
| <BN-Conv-BN-ReLU-Conv-BN-ReLU-Conv-BN-add> | RandomDrop     | 0.05  | 23.89    |           |
|                          | ShakeDrop      | 0.025 | 23.95    |           |
|                          |                |       | 24.06    |           |

5.2. Comparison on ImageNet

We also conducted experiments on ImageNet dataset [17] using ResNet, ResNeXt and PyramidNet. All of the network architectures used in this experiments were of 50 layers. We used small $p_L = 0.025, 0.05$. We recommend a smaller $p_L$ for smaller network architectures. The same observation was obtained in the experimental study about the relationship between $p_L$ of RandomDrop and generalization performance [13]. Note that while [13] reports RandomDrop with $p_L = 0.5$ did not improve the accuracy on a very complex and large dataset such as ImageNet dataset, we found that RandomDrop with small $p_L$ such as $p_L = 0.025, 0.05$ improved the generalization performance.

Table B2 shows the experimental results. [7] reports the standard deviation of the error on ImageNet dataset, which was 0.12%. Taking this into account, ShakeDrop clearly outperformed RandomDrop on ResNet and ResNeXt, and ShakeDrop was comparable to RandomDrop on PyramidNet. Comparing the original and EraseReLU architectures, the original was better for all cases.

6. Conclusion

We proposed a new stochastic regularization method ShakeDrop which, in principle, can be successfully applied to ResNet family as long as the building blocks end with batch normalization. Through the experiments on CIFAR-100 and ImageNet datasets, we confirmed that in most cases, ShakeDrop outperformed existing regularization methods of the same category, which are Shake-Shake and RandomDrop.

Acknowledgement

This work is supported by JST CREST #JPMJCR16E1, JSPS KAKENHI #JP25240028, #JP17H01803, #JP18J15255 and #JP18K19785, JST AIP PRISM #J18Z200418, Artificial Intelligence Research Promotion Foundation, and AWS Cloud Credits for Research Program.

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. arXiv preprint 1709.07634, 2017.

[4] X. Gastaldi. Shake-shake regularization. arXiv preprint arXiv:1705.07485v2, 2017.

[5] X. Gastaldi. Shake-shake regularization of 3-branch residual networks. In Proc. ICLR, 2017.

[6] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In Proc. ICLR, 2015.

[7] P. Goyal, P. Dollr, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: Training imagenet in 1 hour. arXiv preprint 1706.02677, 2017.

[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. arXiv preprint arXiv:1610.02915v4, 2017.

[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] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. arXiv preprint arXiv:1603.09382v3, 2016.

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

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

[15] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

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

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

[18] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In Proc. NIPS, 1992.

[19] Y. Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. Soviet Mathematics Doklady, 27:372–376, 1983.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. Nature, 323:533–536, 1986.

[21] 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., 15(1):1929–1958, 2014.

[22] 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.

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

[24] Y. Tokozume, Y. Ushiku, and T. Harada. Between-class learning for image classification. In Proc. CVPR, 2018.

[25] V. Verma, A. Lamb, C. Beckham, A. Courville, I. Mitliagkas, and Y. Bengio. Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer. arXiv preprint 1806.05236, 2018.

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

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

[28] Y. Yamada, M. Iwamura, and K. Kise. Deep pyramidal residual networks with separated stochastic depth. arXiv preprint arXiv:1612.01230, 2016.

[29] Y. Yamada, M. Iwamura, and K. Kise. Deep pyramidal residual networks with separated stochastic depth. arXiv preprint arXiv:1612.01230, 2016.

[30] S. Zagoruyko and N. Komodakis. Wide residual networks. In Proc. BMVC, 2016.

[31] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In Proc. ICLR, 2017.
A. Experimental Conditions

All networks were trained using back-propagation by stochastic gradient descent with Nesterov accelerated gradient [19] and momentum method [20]. 4 GPUs (on CIFAR-10/100) and 8 GPUs (on ImageNet) were used for learning acceleration; due to parallel processing, different observations of $b_l$, $\alpha$ and $\beta$ are obtained on each GPU. For example, it can happen that on a GPU the $l$-th layer is perturbed, while the layer is not perturbed on other GPUs ($l$ is an arbitrary number). In addition, even if the layer is perturbed on multiple GPUs, different observations of $\alpha$ and $\beta$ are used depending on GPUs. Experimental conditions depending on datasets are described below.

CIFAR-10/100 dataset

Implementations used in the experiments on CIFAR-10/100 dataset [15] were based on the publicly available code of fb.resnet.torch\(^7\). Input images of CIFAR-100 datasets were processed in the following manner. An original image of $32 \times 32$ pixels was color-normalized, followed by horizontally flipped with a 50% probability. Then, it was zero-padded to be $40 \times 40$ pixels and randomly cropped to be an image of $32 \times 32$ pixels.

On PyramidNet, the initial learning rate was set to 0.5 on both CIFAR-10/100 datasets following the version 2 of the PyramidNet paper [9], while they used 0.1 on CIFAR-10 dataset and 0.5 on CIFAR-100 dataset since the version 3 paper.

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. As the filter parameters initializer, “MSRA” [10] was used. In addition, a weight decay of 0.0001, a momentum of 0.9, and a batch size of 256 were used on 8 GPUs. As the linear decay parameter, $p_L = 0.025$ and $p_L = 0.05$ were used. 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 PyramidNet) with the pixel-level update rule.

B. Experiments on Longer Training

Shake-Shake introduced longer training that can be applied to many methods in the learning process as described in Sec. 2. While most of methods related to ResNet use 300-epoch scheduling for learning on CIFAR-10/100 dataset, Shake-Shake uses 1800-epoch cosine annealing, on which the initial learning rate is annealed using a cosine function without restart [4]. As the longer training may contribute to improvement of accuracy, we also applied longer training to the proposed method.

B.1. CIFAR-10 and CIFAR-100

ShakeDrop was compared with Shake-Shake on CIFAR-10/100 datasets on longer training conditions. The ResNeXt experimental condition is following [4]. The ResNet, ResNet (EraseReLU version), and PyramidNet experimental conditions are the same as the CIFAR-100 condition as shown in Sec. A. We used 300-epoch multi-step learning as shown in Sec. A, and 1800-epoch cosine annealing as longer training [4].

Table B1 shows the error rates. “ResNeXt” and “ResNeXt + Shake-Shake” combined with the longer training improved the error rates from the 300-epoch multi-step learning. In addition, “ResNet + ShakeDrop”, “ResNet (EraseReLU) + ShakeDrop”, and “PyramidNet + ShakeDrop” combined with the longer training improved the error rates. Hence, the longer training is an important factor on the Shake-Shake experiments and the longer training is effective for the ShakeDrop regularization.

B.2. ImageNet

ShakeDrop was compared with the vanilla network (without regularization) and RandomDrop on ImageNet classification datasets on longer training conditions. The experimental conditions is following Sec. A. We used 180-epoch multi-step learning as longer training. The initial learning rate was decayed by a factor of 0.1 at 60, 120 and 160 epochs of the entire learning process (180 epochs), respectively.

Table B2 shows the error rates. Almost all networks on the longer training improved the error rates, and ShakeDrop tended to outperform the others on the longer train-

---

\(^7\)https://github.com/facebook/fb.resnet.torch
\(^8\)https://github.com/chainer/chainer/tree/master/examples/imagenet
Table B1. Top-1 errors (%) at the final epoch (300th or 1800th) on the CIFAR-10/100 datasets. Representative methods and the proposed ShakeDrop applied to ResNet, ResNet (EraseReLU), and PyramidNet are compared. In ShakeDrop, $\alpha = 0, \beta \in [0, 1]$ were used in the original ResNet and $\alpha \in [-1, 1], \beta \in [0, 1]$ in the PyramidNet and the “EraseReLU”ed ResNet. “Gain” represents reduced error rates by longer training (i.e., Error rates of “300 Epoch” − Error rates of “1800 Epoch”). Positive value means longer training is effective. * indicates the result is quoted from [4]. + indicates the average result of 4 runs.

(a) CIFAR-10

| Method               | Regularization | Depth | #Param | 300 Epoch | 1800 Epoch | Gain |
|----------------------|----------------|-------|--------|-----------|------------|------|
| ResNeXt [27, 4]      | -              | 26    | 26.2M  | 4.02      | *3.58      | 0.44 |
|                      | Shake-Shake    | 26    | 26.2M  | 3.36      | *2.86      | 0.50 |
| ResNet [11]          | ShakeDrop      | 110   | 1.7M   | 4.78      | *4.08      | 0.70 |
| ResNet (EraseReLU) [3]| ShakeDrop      | 110   | 1.7M   | 5.08      | 3.81       | 1.27 |
| PyramidNet [9]       | ShakeDrop      | 272   | 26.0M  | 3.41      | 2.67       | 0.74 |

(b) CIFAR-100

| Method               | Regularization | Depth | #Param | 300 Epoch | 1800 Epoch | Gain |
|----------------------|----------------|-------|--------|-----------|------------|------|
| ResNeXt [27, 4]      | -              | 29    | 34.4M  | 16.95     | *16.34     | 0.61 |
|                      | Shake-Shake    | 29    | 34.4M  | 16.46     | *15.85     | 0.61 |
| ResNet [11]          | ShakeDrop      | 110   | 1.7M   | 23.74     | 21.18      | 2.56 |
| ResNet (EraseReLU) [3]| ShakeDrop      | 110   | 1.7M   | 21.81     | 19.81      | 2.00 |
| PyramidNet [9]       | ShakeDrop      | 272   | 26.0M  | 14.90     | 13.99      | 0.91 |

Table B2. Top-1 errors (%) on ImageNet dataset at 90th and 180th epochs. This “Type” indicates either the original or “EraseReLU”ed networks. In ShakeDrop, $\alpha = 0, \beta \in [0, 1]$ were used in the original ResNet and the original ResNeXt, and $\alpha \in [-1, 1], \beta \in [0, 1]$ in the original PyramidNet and the “EraseReLU”ed networks. “Gain” represents reduced error rates by longer training (i.e., Error rates of “90 Epoch” − Error rates of “180 Epoch”). Positive value means longer training is effective.

| Methods           | Type       | Regularization | $p_L$ | 90 Epoch | 180 Epoch | Gain |
|-------------------|------------|----------------|-------|----------|-----------|------|
| **ResNet-50**     | Original   | Vanilla        | -     | 23.60    | 23.20     | 0.40 |
|                   |            | RandomDrop     | 0.05  | 23.36    | 22.66     | 0.70 |
|                   |            | ShakeDrop      | 0.05  | 23.33    | *22.52    | 0.81 |
|                   | EraseReLU  | Vanilla        | -     | 25.63    | 25.03     | 0.60 |
|                   |            | RandomDrop     | 0.05  | 25.61    | 24.77     | 0.84 |
|                   |            | ShakeDrop      | 0.05  | 25.69    | *24.53    | 1.16 |
| **ResNeXt-50 32-4d** | Original  | Vanilla        | -     | 22.78    | 22.83     | -0.05|
|                   |            | RandomDrop     | 0.05  | 22.47    | 22.68     | -0.21|
|                   |            | ShakeDrop      | 0.05  | 22.38    | *22.34    | 0.04 |
|                   | EraseReLU  | Vanilla        | -     | 24.66    | 24.33     | 0.33 |
|                   |            | RandomDrop     | 0.05  | 24.45    | 23.97     | 0.48 |
|                   |            | ShakeDrop      | 0.05  | 24.41    | *23.64    | 0.77 |
| **PyramidNet-50 $\alpha=300$** | Original | Vanilla        | -     | 24.06    | 23.73     | 0.33 |
|                   |            | RandomDrop     | 0.05  | 23.89    | *23.24    | 0.65 |
|                   |            | ShakeDrop      | 0.05  | 23.95    | 23.33     | 0.62 |

Hence, the combination of ShakeDrop and the longer training is effective for improvement of the generalization performance.