Regularizing Neural Networks via Stochastic Branch Layers

Wonpyo Park[1,4*] PARKWONPYO@POSTECH.AC.KR
Paul Hongsuck Seo[1,2*] HSSEO@POSTECH.AC.KR
Bohyung Han[2,3] BHHAN@SNU.AC.KR
Minsu Cho[1,3] MSCHO@POSTECH.AC.KR
[1] *Computer Vision Lab, CSE, POSTECH*
[2] *Computer Vision Lab, ECE & ASRI, Seoul National University*
[3] *The Neural Processing Research Center*
[4] *Kakao Corporation*

Editors: Wee Sun Lee and Taiji Suzuki

Abstract

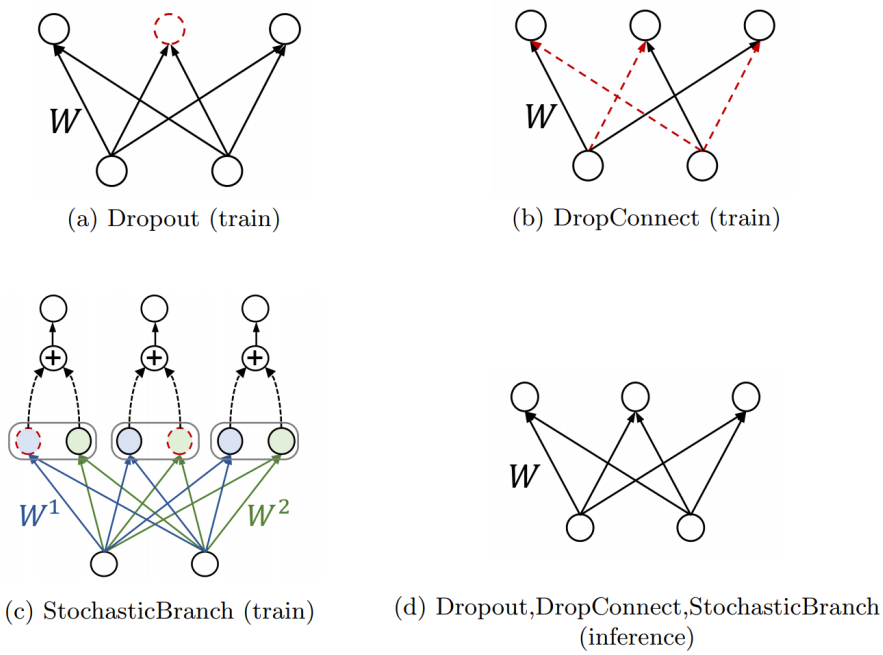
We introduce a novel stochastic regularization technique for deep neural networks, which decomposes a layer into multiple branches with different parameters and merges stochastically sampled combinations of the outputs from the branches during training. Since the factorized branches can collapse into a single branch through a linear operation, inference requires no additional complexity compared to the ordinary layers. The proposed regularization method, referred to as StochasticBranch, is applicable to any linear layers such as fully-connected or convolution layers. The proposed regularizer allows the model to explore diverse regions of the model parameter space via multiple combinations of branches to find better local minima. An extensive set of experiments shows that our method effectively regularizes networks and further improves the generalization performance when used together with other existing regularization techniques.

1. Introduction

Deep neural networks have made a remarkable progress in a variety of fields including computer vision, natural language processing, medical imaging, speech recognition, and computer graphics. Since many tasks in the fields require to understand high-level semantics, neural networks tend to go deeper and over-parametrized. Such deep and large networks are prone to overfitting so that a proper regularization becomes a critical factor in improving their generalization performance. A popular type of regularization for deep neural networks is to inject random noise into the networks during training, *e.g.*, applying a binary random mask to hidden activations (Hinton et al., 2012) or weights (Wan et al., 2013), or skipping layers (Huang et al., 2016) by forwarding activations via random identity connections. Due to its simplicity and effectiveness, the stochastic regularization is widely used for training deep neural networks.

We propose a novel regularization technique referred to as *StochasticBranch*, which decomposes an ordinary linear layer into the one with multiple stochastic branches. By factorizing the original weight matrix of the layer into a set of matrices with their random

. *Equal contribution



(a) Dropout (train)

(b) DropConnect (train)

(c) StochasticBranch (train)

(d) Dropout,DropConnect,StochasticBranch (inference)

Figure 1: StochasticBranch compared to Dropout and DropConnect. These stochastic regularizers introduce the binary random mask during training. Red dashed lines represent what is masked out. Dropout and DropConnect apply random masks to activations and connections, respectively. StochasticBranch first decomposes each linear unit into multiple branches (two branches colored in blue and green) and injects random masks on the outputs of branches during training. At inference time, these branches are merge back to a single branch by taking the expectation of activations. Note that the inference procedures of Dropout, Dropconnect and StochasticBranch are exactly same and require the identical computational cost. StochasticBranch can be interpreted as the generalization of Dropout and DropConnect.

binary masks, the stochastic branches effectively regularize a network during training. As a generalization of Dropout, its rich ensemble property with decomposed models allows to investigate exponentially many distinct models during training and explore diverse regions of the parameter space resulting in the better local optima. At inference time, the multiple branches collapse back into a single branch, thus requiring no additional complexity compared to the normal linear layers. Fig. 1 illustrates the comparison to Dropout (Hinton et al., 2012) and Dropconnect (Wan et al., 2013). An extensive set of experiments shows the effectiveness of the proposed technique as well as wide applicability together with other popular regularizers including Dropout (Hinton et al., 2012) and Batch Normalization (Ioffe and Szegedy, 2015).

2. Related Work

Regularization is a common and essential technique to combat overfitting in training. While one common form of the techniques is to penalize the weight tensor with a constant (Krogh and Hertz, 1992; Srebro and Shraibman, 2005), a popular method for deep neural networks is to inject random noise during training. The most well-known example is Dropout (Hinton et al., 2012) which stochastically zero out activations of neural networks to avoid co-adaptation of neurons. Several successive follow-ups of Dropout have been proposed. Wan et al. (2013) propose a generalization of Dropout, called DropConnect, which zero-out weight rather than activation. Ba and Frey (2013) propose adaptive Dropout, where drop rate of activation is determined by a binary belief network overlaid on the neural network. Li et al. (2016) introduce an efficient evolutionary Dropout that computes sampling probabilities on-the-fly from a mini-batch of examples. Bulò et al. (2016) develop Dropout distillation for better approximating the average predictor without sacrificing the computational efficiency of standard Dropout. Kang et al. (2016) propose Shakeout that randomly enhances or inverses contributions of each unit to the next layer, resulting in combination of L1 and L2 regularization. Gal and Ghahramani (2016) introduce a theoretical framework for casting dropout as Bayesian inference to approximate uncertainty of neural networks. Zhai and Wang (2018) propose a framework to adaptively adjust the drop rates based on the Rademacher complexity bound.

Other types of regularizers using stochastic noise have also been proposed to further improve generalization. Zeiler and Fergus (2013) introduce a stochastic pooling that randomly picks activation within each pooling region according to a multinomial distribution. Smith et al. (2016) develop a dynamically growing neural network, Dropin, that gradually decreases the probability of skipping layers to train a network from shallow layers to deeper layers. Huang et al. (2016) propose to train a very deep ResNet network by stochastically skipping ResNet blocks via identity connections. Ma et al. (2016) introduce expectation-linear Dropout that regularizes the training objective with a measured inference gap. Noh et al. (2017) reduce the gap between a marginal likelihood and a training objective with stochastic noise injection.

There exist recent methods that leverage multiple branches to improve generalization performance. Lee et al. (2015) propose a multi-head ensemble learning that shares early convolutional layers. Han et al. (2017) introduce a regularized ensemble method for single object tracking, which branches out intermediate layers to learn different target representations. Unlike our method, these approaches aim at proposing a specific type of architectures for ensemble learning rather than a generic regularization method for neural networks. Goodfellow et al. (2013) propose the Maxout activation function that merges outputs of branches of a single layer by max-pooling. It only helps the model with a stochastic regularizer better approximate ensemble results by model averaging, whereas our method itself is an effective stochastic regularizer.

3. Stochastic Branch

This section presents the details of StochasticBranch and relates the method to Dropout (Hinton et al., 2012; Srivastava et al., 2014). For ease of explanation, we will only discuss a

fully-connected (`fc`) layer in this section. Note, however, that StochasticBranch is applicable to any type of linear operators including convolution.

3.1. Stochastic Branch Layer

Let us consider an `fc` layer with input $\mathbf{x} \in \mathcal{R}^D$ and output $\mathbf{y} \in \mathcal{R}^{\hat{D}}$:

$$\mathbf{y} = \sigma\left(\mathbf{W}\mathbf{x}\right) \qquad \text{and} \qquad y_i = \sigma\left(\sum_{j=1}^{D} w_{i,j} x_j\right), \tag{1}$$

where $\mathbf{W}$ is a weight matrix and $\sigma(\cdot)$ is an element-wise nonlinear activation function such as ReLU and tanh.

We decompose the weight matrix $\mathbf{W}$ into a sum of N matrices, *i.e.*, , $\mathbf{W} = \sum_{k=1}^{N} \mathbf{W}^k$, such that the pre-activation for each output unit is given by the sum of N linear projections:

$$y_i = \sigma\left(\sum_{j=1}^{D} w_{i,j} x_j\right) = \sigma\left(\sum_{j=1}^{D}\sum_{k=1}^{N} w_{i,j}^k x_j\right). \tag{2}$$

This network structure can be interpreted as integrating multiple branches, where an output node is computed from the sum of N branches.

In order to make the branches stochastic, we now introduce a random variable, $m_i^k \sim \text{Bernoulli}(p^k)$, to each branch in Eq. (2), thus resulting in

$$y_i = \sigma\left(\sum_{j=1}^{D}\sum_{k=1}^{N} m_i^k w_{i,j}^k x_j\right) = \sigma\left(\sum_{k=1}^{N} m_i^k \sum_{j=1}^{D} w_{i,j}^k x_j\right). \tag{3}$$

As shown in Figure 1, each output node of the stochastic branch layer is obtained using N stochastic branch units. All input nodes of the layer are connected to the branch units, producing N distinct values. The random binary masks then zero out a subset of the values, and the corresponding output activation is computed from the sum of the masked values.

Any linear layer parametrized by a weight matrix $\boldsymbol{W}$ can be transformed to a StochasticBranch layer with N branches, *e.g.*, by setting the weight matrix of k-th branch to $\boldsymbol{W}^k (k = 1, ..., N)$ where the sum of those weights is $\boldsymbol{W}$. In training a neural network, the stochastic branch layers act as a regularizer. A set of $\hat{D}N$ random binary masks $\{m_i^k\}$ is sampled for each training example and used in both forward and backward passes. To update the weight matrices in the layer, *e.g.*, via stochastic gradient descent (SGD), only the branches that were active in the forward pass are updated. Note that this stochastic training procedure induces the branches to be distinctive from each other. We present the effect in Section 4.

At inference time, instead of sampling random masks, we compute the output y_i^* by taking the expectation of pre-activations using the following procedure:

$$\begin{aligned} y_i^* = \sigma\left(\mathbb{E}[y_i]\right) &= \sigma\left(\mathbb{E}\left[\sum_{k=1}^{N} m_i^k \sum_{j=1}^{D} w_{i,j}^k x_j\right]\right) \\ &= \sigma\left(\sum_{k=1}^{N} \mathbb{E}\left[m_i^k\right] \sum_{j=1}^{D} w_{i,j}^k x_j\right) \\ &= \sigma\left(\sum_{k=1}^{N} p^k \sum_{j=1}^{D} w_{i,j}^k x_j\right) \\ &= \sigma\left(\sum_{j=1}^{D} \hat{w}_{i,j} x_j\right), \end{aligned} \tag{4}$$

where

$$\hat{w}_{i,j} = \sum_{k=1}^{N} p^k w_{i,j}^k.$$

Note that the multiple units branched for training now merge back into a single unit; there is no additional computational cost for inference compared to an ordinary `fc` layer as shown in Figure 1d.

3.2. Generalized Dropout

StochasticBranch is a generalization of Dropout (Hinton et al., 2012; Srivastava et al., 2014) and DropConnect (Wan et al., 2013), which can be shown below by imposing additional constraints on the StochasticBranch formulation.

If we impose a group masking constraint that an identical mask m_i is used for all branches with the same output unit y_i, *i.e.*, $m_i^k = m_i$ for all k, the multiple branches of Eq. (3) collapse into a single branch with a random mask variable:

$$y_i = \sigma\left(\sum_{k=1}^{N} m_i \sum_{j=1}^{D} w_{i,j}^k x_j\right) = \sigma\left(m_i \sum_{j=1}^{D} w_{i,j} x_j\right). \tag{5}$$

For any zero-centered activation function σ such as ReLU and tanh, we can move the mask variable m_i out of the activation function so that it becomes equivalent to the Dropout regularizer (Hinton et al., 2012; Srivastava et al., 2014):

$$y_i = m_i \sigma\left(\sum_{j=1}^{D} w_{i,j} x_j\right). \tag{6}$$

This shows that Dropout is StochasticBranch under the constraint of group masks. Dropout either removes or retains an entire activation, whereas StochasticBranch rejects parts of the activation by masking out a subset of decomposed weights $\mathbf{W}^k$ in multiple branches.

If we impose a one-to-one branching constraint that each input x_j is paired with exactly one branch, each input-output connection $w_{i,j}x_j$ involves a mask variable. For example, consider D branches where $w_{ij}^k = w_{ij}$ for $k = j$ and $w_{ij}^k = 0$ for $k \neq j$. Then, StochasticBranch of Eq. (3) reduces to

$$y_i = \sigma\left(\sum_{k=1}^{D} m_i^k \sum_{j=1}^{D} w_{i,j}^k x_j\right) = \sigma\left(\sum_{k=1}^{D} m_i^k w_{i,k} x_k\right), \tag{7}$$

which is exactly the same form with another generalized Dropout called DropConnect (Wan et al., 2013):

$$y_i = \sigma\left(\sum_{j=1}^{D} m_{i,j} w_{i,j} x_j\right). \tag{8}$$

This in turn shows that DropConnect is StochasticBranch under the constraint of one-to-one branches. DropConnect (Wan et al., 2013) sample each input-output connection, whereas StochasticBranch maintains different weights across multiple branches and sample a branch, rather than a connection, for each unit.

As a generalization of Dropout and DropConnect, StochasticBranch plays the role of a strong stochastic regularizer as will be discussed in the following subsection. Note, however, that a combination with other methods is also possible and may become a better regularizer as an extension. For example, if the random mask of Dropout is added to StochasticBranch, the combination can be represented as

$$y_i = m_i \sigma\left(\sum_{k=1}^{N} m_i^k \sum_{j=1}^{D} w_{i,j}^k x_j\right). \tag{9}$$

which is a further generalization of StochasticBranch with additional group masking m_i of Dropout. Dropout is designed to mitigate the problem of co-adaptation that neurons excessively rely on other neurons (Hinton et al., 2012; Srivastava et al., 2014). By turning off neurons with probability of $1 - p$, Dropout encourages neurons to less co-adapt each other and induces the layer to produce sparse activations. In contrast, StochasticBranch activates neurons via 2^N combinations of branch outputs, and induces the layer to produce diverse activations across examples. And, its turn-off chance is significantly smaller than that of Dropout, which is probability of $\prod_{k=1}^{N}(1 - p^k)$. Considering the differences, the two techniques may complement each other in practice. We will demonstrate such combination effects in Section 4.

3.3. Discussion

Ensemble learning. From an ensemble learning point of view (Hinton et al., 2012; Baldi and Sadowski, 2014), Dropout and its generalizations can be interpreted as learning an exponentially large ensemble of networks, where each model of the ensemble is given training examples in different orders via mini-batching during training. Each method approximates an ensemble from different classes of networks. Previous methods such as Dropout and

DropConnect draw such models only within the original neural network. For example, Dropout approximates geometric ensemble averaging of $2^{\hat{D}}$ models (Baldi and Sadowski, 2014) where $\hat{D}$ denotes the number of units with Dropout. In contrast, StochasticBranch draws models from a richer class of networks augmented by branching so that the models in the ensemble are parameterized with different weights from distinct combinations of branches. It thus approximates an ensemble of $2^{\hat{D}N}$ models where $\hat{D}$ and N is the number of StochasticBranch units and branches, respectively. This rich ensemble with decomposed models allows to explore different regions of the parameter space and may find a diverse set of local optima.

Data augmentation. Dropout can also be seen as an implicit form of sophisticated data augmentation increasing training data coverage (Konda et al., 2015), *e.g.*, in case of images, translations, rotations, scaling, etc. Noise induced by a random mask $\mathbf{m}$ results in a similar effect of augmenting data $\mathbf{x}$ using a set of such transformations $f \in \mathcal{F}$ so that $\mathbf{m} \odot \sigma(\mathbf{W}\mathbf{x}) \approx \sigma(\mathbf{W}f(\mathbf{x}))$ in the case of a single layer model. Here, a model in the Dropout ensemble can correspond to a transformation f for data augmentation. In this perspective, StochasticBranch creates a larger set of fine-grained transformations by decomposing transformations of Dropout, resulting in an effect of richer data augmentation.

Batch normalization. It has been widely known that Dropout and Batch Normalization are in disharmony each other in using them together. A recent research (Li et al., 2018) shows that a cause of the disharmony is a variance shift of activations in Dropout between training and testing, and suggests to reduce the variance shift by placing Batch Normalization before random noise injection of Dropout. For the same reason, when Batch Normalization is used together with StochasticBranch, we place Batch Normalization before noise injection of StochasticBranch. Compared to Dropout, we observe that StochasticBranch has a lower variance shift, thus being more compatible with Batch Normalization. The corresponding experiments are reported in Section 4.1.

Maxout. Maxout networks (Goodfellow et al., 2013) also have a similar branching structure where a layer merges outputs of multiple branches. Despite its apparent similarity to StochasticBranch, its goal and structure are significantly different from ours. Maxout is a non-linear activation function by max-pooling that is designed to improve ensemble approximation with stochastic regularizers by model averaging. It can thus also be used together with the proposed stochastic regularizer. Moreover, all branches of Maxout networks need to maintain their parameters even at inference due to the max-pooling operation, whereas multiple branches of StochasticBranch merge back into a single unit at inference.

Time Complexity. SB introduces additional time complexity within only a few layers where SB is applied. Therefore, the increase of overall complexity is not significant in most cases. For instance, the use of SB on ResNet-110 in our experiments (refer to Table 4) increases only 3.62% of time complexity (2.48G vs. 2.57G flops). Note that SB increases time complexity only during training while the inference complexity remains the same as ordinary networks.

Table 1: Averaged classification error [%] and standard deviation on MNIST and FMNIST with five runs.

	MNIST (Error/stdev)			FMNIST (Error/stdev)		
	MLP3	MLP5	CNN	MLP3	MLP5	CNN
Baseline	1.79 / 0.06	1.95 / 0.11	0.88 / 0.02	10.08 / 0.16	10.12 / 0.16	8.34 / 0.15
+DO	1.46 / 0.03	1.72 / 0.07	0.68 / 0.03	9.44 / 0.05	9.96 / 0.19	7.65 / 0.17
+BN	1.54 / 0.06	1.58 / 0.06	0.74 / 0.05	10.04 / 0.20	9.68 / 0.16	9.65 / 0.13
+DO+BN	1.42 / 0.05	1.42 / 0.04	0.74 / 0.05	9.37 / 0.12	9.55 / 0.18	9.05 / 0.13
+SB	1.47 / 0.03	1.55 / 0.05	0.73 / 0.04	9.60 / 0.14	9.64 / 0.03	8.04 / 0.17
+SB+DO	1.30 / 0.02	1.34 / 0.03	0.63 / 0.03	**9.18** / 0.08	9.52 / 0.04	7.66 / 0.06
+SB+BN	**1.25** / 0.03	**1.19** / 0.02	**0.45** / 0.03	9.25 / 0.09	**9.19** / 0.16	**7.36** / 0.20

4. Experiments

We evaluate StochasticBranch on multiple image classification benchmarks. In the experiments, our method (SB) is compared with two of the most popular regularization methods: Dropout (DO) and Batch Normalization (BN). We set the drop rates of SB and DO to 0.5, and use 10 branches ($N = 10$) unless specified otherwise.

4.1. MNIST and Fashion-MNIST

We first conduct a set of experiments on MNIST (LeCun et al., 1998) and Fashoin-MNIST (FMNIST) (Xiao et al., 2017). Both benchmarks consist of 28×28 grayscale images with 10 class labels. MNIST classes represent digits between 0 and 9, whereas FMNIST classes correspond to fashion items. In both benchmarks, training and test sets contain 60,000 and 10,000 examples, respectively. We test our method on multi-layer perceptrons (MLP) and convolutional neural networks (CNN). We implement two MLPs with 3 and 5 layers (MLP3 and MLP5) where each intermediate layer has 1,024 hidden units. CNN consists of two convolution (`conv`) layers with 5×5 kernels and two `fc` layers. The output channels of the first and second `conv` layers are set as 32 and 64, respectively, while the number of hidden activations of the first `fc` layer is 1,024. In every network, we use ReLU function for intermediate activations. We train the three models without any regularization techniques as our baselines to reveal the improvements by regularization methods. SB is applied to every layer of MLP3 and CNN. For MLP5, we apply the technique to the first, third and fifth layers. BN is applied to pre-activations of every layer and DO is placed in between every successive `fc` layers.

Table 1 summarizes classification errors of the models with different regularization techniques on MNIST and FMNIST, where all results are obtained by averaging the errors of five independent runs. The proposed method reduces the baseline errors in all settings comparable to or often better than other techniques. Notably, our regularizer further reduces the errors when combined with other regularizers, achieving the largest error reductions in all settings. Interestingly, Table 1 reveals that the error reduction of SB with BN is significantly larger than that of DO with BN. This is because SB has a lower variance shift

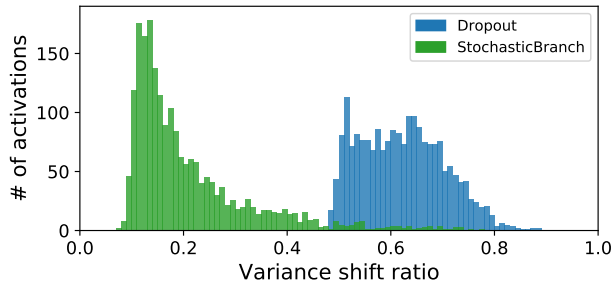


Figure 2: Histogram of variance shift ratio of activation after Dropout and StochasticBranch.

compared to DO. To quantify variance shifts of SB and DO, we measure variance shift ratio (VSR),which is given by

$$\text{VSR} = \frac{\text{VAR}(y) - \text{VAR}(y^*)}{\text{VAR}(y)} \tag{10}$$

where $\text{VAR}(y)$ and $\text{VAR}(y^*)$ represent the activation variances for each neuron at training and test time, respectively.

Note that the ratio of 0 is the ideal case where there is no variance shift. Figure 2 presents VSR distribution of hidden activations of MLP3 on MNIST test set. The figure clearly shows that activations of SB has lower variance shifts than DO. In average, SB has VSR of 0.21 while DO shows that of 0.62.

In addition, we conduct experiments with Maxout to show that it is distinct from and complementary to StochasticBranch as discussed in Section 3.3. We test DO and SB models of MLP3 with Maxout on MNIST by replacing the non-linearity function of the stochastic layers. The use of Maxout further reduces the classification error by 0.04% and 0.14% for DO and SB, respectively. Note that the additional error reduction by Maxout is larger with SB than with DO.

Figure 3 shows the effects of SB, DO, and SB+DO in terms of activation statistics on MNIST test set. The statistics are measured with the activations at the second `fc` layer of MLP3. We present two histograms for each method: (1) histogram of mean activation of each neuron and (2) histogram of the number of active neurons for each image. A neuron with zero mean activation is a dead neuron, and an image with a small number of active neurons corresponds to a case with sparse representations. Note that if the weights of a neuron converge to a point where its preactivation is severely biased to a negative value, the neuron may become near-dead and almost never activates (Maas et al., 2013). The comparison between the baseline (Figure 3a) and DO (Figure 3b) shows that DO reduces near-dead neurons as well as excessively-active neurons, and also induces sparse activations. The similar effects have been observed in the work of (Srivastava et al., 2014). Interestingly, Figure 3c shows that SB is significantly more effective in reducing near-dead neurons than DO. Since the process of stochastic branching generates exponentially many weight combinations, some combinations without negative bias may allow dead neurons to activate again by receiving the gradient during training. Note that dead-neurons with ReLU is not able to be active again in both the baseline and DO since the dead-neurons never receive the gradient signal during training. As a side effect, the input features become denser as indicated by a large

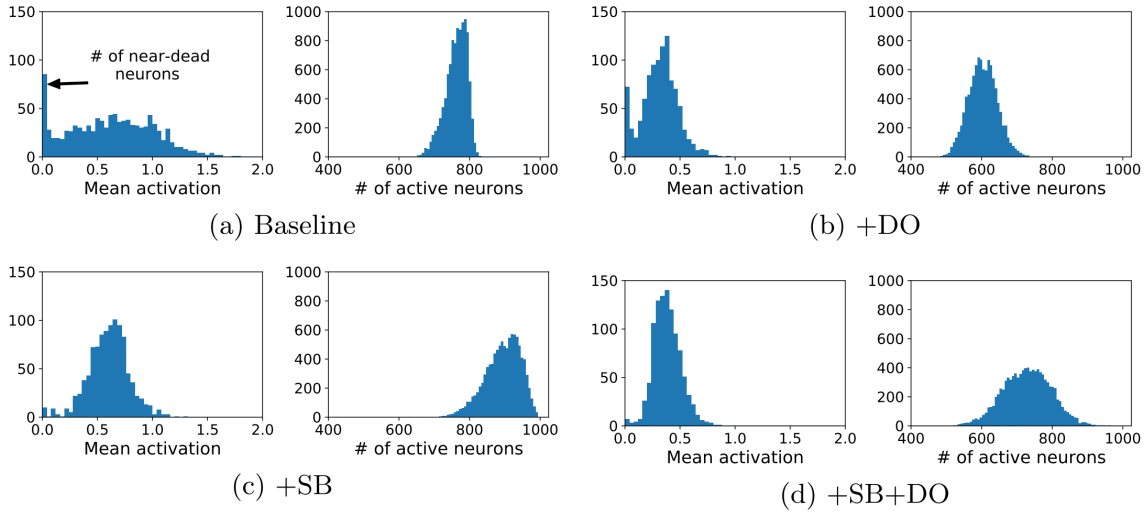
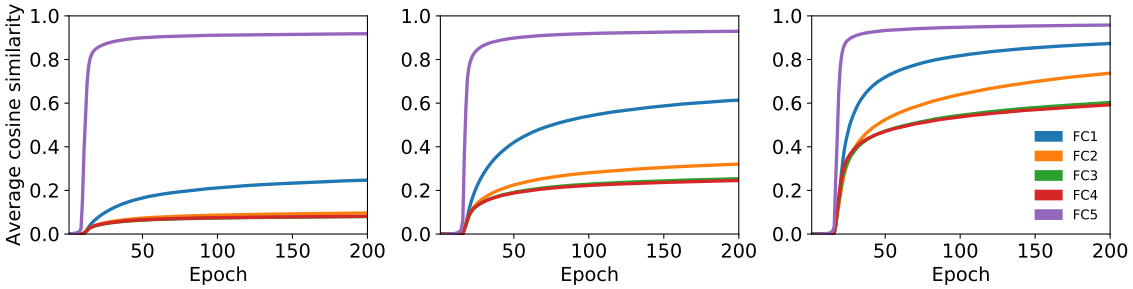


Figure 3: Effects of StochasticBranch, Dropout, and StochasticBranch+Dropout on MNIST test set. We present two histograms for each model: (left) the number of neurons vs. the mean activation value of the neuron over all images, (right) the number of images vs. the number of active neurons per image. Note that a neuron with mean activation value 0 is a dead neuron, and also that an image with a small number of active neurons means a case with sparse activations.

number of active neurons per image. Due to these two different aspects (DO encouraging sparse representation and SB focusing more on reducing dead neurons), SB and DO may be complementary with each other. Figure 3d demonstrates that the combination of SB+DO lowers the number of active neurons per image and retains sparse activations while reducing near-dead neurons.

Figure 4 shows average cosine similarity of weight vectors between different branches, which is measured over epochs. For this experiment, we train three instances of MLP5 with different number of hidden units (64, 256, 1,024) applying SB to all layers. We observe significantly low cosine similarities between weights of branches in Figure 4a; this confirms that the branches of SB learn distinctive patterns and are capable of exploring various regions of the parameter space. As we decrease the number of hidden units in the layers as in Figure 4b and 4c, the branches become more similar because it is difficult to decompose a pattern with a small number of output units into diverse yet useful patterns. The redundant patterns across branches in these small networks bring lower regularization performance, resulting in relatively smaller gains. For example, the accuracy gain (0.17%) of the model with 64 activations is smaller than those (0.64% and 0.35%) of the models with 1024 and 256 activations. This implies that SB may perform better in layers with more output neurons. Another observation is that the average similarity between branches tends to decrease when SB is applied to deeper layers. The branches at the last layer (`fc5`) have particularly high similarity in all three settings since the last layer has only 10 output units corresponding to the number of classes.



(a) # of activations: 1024 (b) # of activations: 256 (c) # of activations: 64

Figure 4: (a-c) Average pairwise cosine similarity between branches varying number of hidden units in the network. Cosine similarity of weight vectors for an activation in different branches is measured and averaged. We measure the similarity from every layer and observe the difference at different layers.

4.2. CIFAR-10 and CIFAR-100

To validate the proposed method on more realistic settings, we conduct more experiments on CIFAR-10 and CIFAR-100 (Krizhevsky and Hinton, 2009), which contain 32×32 images with 10 and 100 object classes, respectively. The sizes of training and test split are 50,000 and 10,000. For experiments on CIFAR-10, we build a custom CNN that consists of two 5×5 `conv`, one maxpool, two 5×5 `conv`, one maxpool, and two `fc` layers. In this experiment, SB and DO are applied to `fc` layers. BN is applied to all `conv` layers as well as `fc` layers, which produces a larger performance gain of BN. Table 2 summarizes the classification errors of the models on CIFAR-10, which show the similar tendency as on MNIST and FMNIST. To study the effect of varying the number of branches, we also train another set of networks whose the first `fc` layers are replaced by SB layers with 2, 4 or 8 branches.

We plot the curves of training loss and the test accuracy in Figure 5a and 5b, respectively. While SB injects stochastic noises diversifying hidden units, we notice that more branches help the network converge better compared to SB with fewer branches. More branches create a richer ensemble of SB during training to better exploration of the parameter space, resulting in a higher chance to converge faster.

To show the richer data augmentation effect of SB discussed in Section 3.3, we conduct additional experiments on CIFAR-10 with fewer training examples. We train models with 1% of randomly sampled training examples for each class. Table 3 shows the classification errors in this setting. As shown in Table 3, SB reduces the error more significantly than DO does where the relative error reduction of DO is 1.7% whereas that of SB and SB+DO are 7.5% and 9.0%, respectively. These results imply that SB has a stronger data augmentation effect compared to DO.

For CIFAR-100, we train two advanced convolutional neural network architectures, ResNet-110 (He et al., 2016) and MobileNetV2 (Sandler et al., 2018). For ResNet-110, both DO and SB are applied on the third `conv` layer of the last bottleneck block. For MobileNetV2, both DO and SB are applied on the depthwise `conv` layer of the last two inverted residual

Table 2: Classification errors of models with different regularization techniques on CIFAR-10.

	Error/stdev
Baseline	14.53 / 0.14
+DO	13.92 / 0.23
+BN	11.99 / 0.22
+DO+BN	12.00 / 0.08
+SB	13.84 / 0.14
+SB+DO	13.82 / 0.24
+SB+BN	**11.83** / 0.29

Table 3: Classification error [%] on CIFAR-10 with 1% of training images.

	Baseline	+DO	+SB	+SB+DO
Error	60.68	59.63	56.14	**55.24**

blocks. The results are summarized in Table 4. While DO performs similarly to the baseline, SB outperforms these models in both networks. These results show the effectiveness of SB in `conv` layers. It is worth noting that SB is more effective for layers with a large number of channels or a large kernel size. It is the reason why SB is used to the `conv` layers in the last blocks in these experiments.

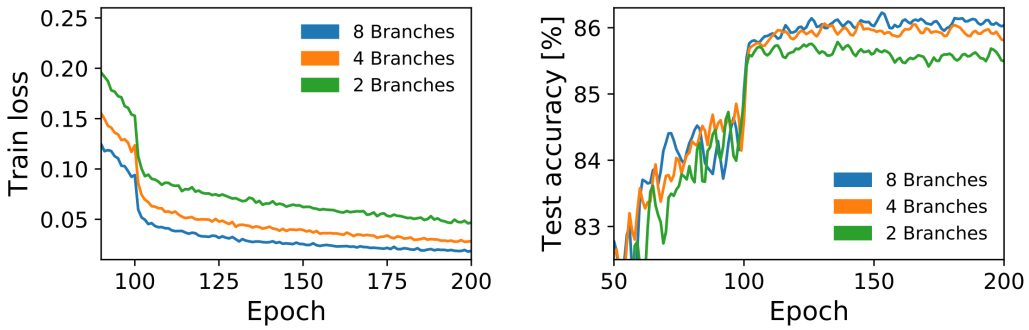
4.3. PASCAL VOC

In following experiments, we apply SB to pretrained networks . To apply SB on a pretrained layer, We initialize the weight matrix of the branches by $\boldsymbol{W}^k = \frac{1}{Np^k}\boldsymbol{W}$ where $\boldsymbol{W}$ is the pretrained weight matrix. Note that although the weight matrices $\boldsymbol{W}^k$ are initialized by scaling the same pretrained weights $\boldsymbol{W}$, the stochastic branch layer is still capable of regularizing the network since each branch observes different training samples, and in consequence, the weight vectors of different branches become dissimilar with each other.

We conduct experiments on multiple tasks using PASCAL VOC (Everingham et al., 2010) benchmarks: multi-label object classification, object detection, and semantic segmentation.

Table 4: Classification error [%] on CIFAR-100 tested with ResNet-110 and MobileNetV2 with single `fc` layer. We apply regularization techniques on `conv` layers.

	Error/stdev
ResNet-110 (Baseline)	24.29 / 0.19
ResNet-110 +DO	24.28 / 0.97
ResNet-110 +SB	**23.41** / 0.34
MobileNetV2 (Baseline)	27.82 / 0.39
MobileNetV2 +DO	27.63 / 0.25
MobileNetV2 +SB	**27.16** / 0.32



(a) Training loss curve (b) Test accuracy curve

Figure 5: Results on CIFAR-10: (a) Training loss curves and (b) test accuracy curves of models with different number of branches in SB layer. For clearance, we smooth test accuracy curves with window size of 5.

Table 5: Mean average precision [%] of classification on PASCAL VOC 2012 validation and test sets.

	Validation mAP/stdev	Test mAP
Baseline	84.47 / 0.09	84.27
+DO	85.54 / 0.11	85.38
+SB	86.19 / 0.02	86.04
+SB+DO	**86.50** / 0.03	**86.36**

For evaluation metrics, mean average precision (mAP) is used for classification and detection while mean intersection over union (mIoU) is for semantic segmentation. For all the tasks, ImageNet-pretrained VGG-16 (Simonyan and Zisserman, 2014) are used as our backbone networks, and SB is applied to `fc7` and DO is placed after the non-linear activations of `fc6` and `fc7`.

For object classification, we finetune the pretrained network on the training set after replacing the classification layer (`fc8`) to match the number of classes in PASCAL VOC; the trained models are evaluated on both validation and test sets. Table 5 shows mAPs of the multi-label classification models on PASCAL VOC 2012. As seen in other experiments where we train networks from scratch, our method effectively improves the baseline performance by regularizing the network even though the weight vectors of branches are equally initialized.

For object detection and semantic segmentation tasks, which require are more complex and structured prediction, we use Faster-RCNN (Ren et al., 2015) and FCN-32S (Long et al., 2015), respectively, as the original architectures are equipped with DO as a regularizer. Table 6 shows mAPs of the object detection models that are trained using training and validation sets of PASCAL VOC 2007 and evaluated on test set. SB effectively regularizes the object detection network and outperforms both baseline and DO. Note that Faster-RCNN architecture not only predicts class labels but also regresses bounding boxes. Table 7 presents

Table 6: Mean average precision [%] of object detection on PASCAL VOC 2007 test set.

	Baseline	+DO	+SB
mAP	70.17	70.36	**71.41**

Table 7: Mean intersection over union [%] of semantic segmentation on PASCAL VOC 2011 validation set.

	Baseline	+DO	+SB
mIoU	62.72	62.84	**63.34**

the results of PASCAL VOC semantic segmentation task. It also shows that SB is also effective for semantic segmentation resulting in an improved mIoU.

5. Conclusion

In this paper, we proposed a novel regularization technique called StochasticBranch, which is a generalization of Dropout. During training, the proposed method factorizes a single layer into multiple branches and sums up the outputs of the branches after random masking by binary noises. At inference time, the multiple branches are merged back into a single layer. We investigated that the proposed method regularizes the various neural networks on multiple benchmarks successfully and achieves significant improvement over the baseline performances. Moreover, a set of experimental results show that our method can be applied with other commonly used regularizers such as Dropout or batch normalization achieving even further performance improvement.

Acknowledgments

This work is supported by Samsung Advanced Institute of Technology and by Basic Science Research Program (NRF-2017R1E1A1A01077999) through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT.

References

Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

Pierre Baldi and Peter Sadowski. The dropout learning algorithm. *Artificial intelligence*, 210:78–122, 2014.

Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Dropout distillation. In *International Conference on Machine Learning*, pages 99–107, 2016.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2): 303–338, June 2010.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Bohyung Han, Jack Sim, and Hartwig Adam. Branchout: Regularization for online ensemble tracking with convolutional neural networks. In *Proceedings of IEEE International Conference on Computer Vision*, pages 2217–2224, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, pages 448–456. JMLR.org, 2015.

Guoliang Kang, Jun Li, and Dacheng Tao. Shakeout: A new regularized deep neural network training scheme. In *AAAI*, pages 1751–1757, 2016.

Kishore Konda, Xavier Bouthillier, Roland Memisevic, and Pascal Vincent. Dropout as data augmentation. *stat*, 1050:29, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan-Kaufmann, 1992.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.

Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *arXiv preprint arXiv:1801.05134*, 2018.

Zhe Li, Boqing Gong, and Tianbao Yang. Improved dropout for shallow and deep learning. In *Advances in Neural Information Processing Systems*, pages 2523–2531, 2016.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

Xuezhe Ma, Yingkai Gao, Zhiting Hu, Yaoliang Yu, Yuntian Deng, and Eduard Hovy. Dropout with expectation-linear regularization. *arXiv preprint arXiv:1609.08017*, 2016.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 2, 2013.

Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. Regularizing deep neural networks by noise: Its interpretation and optimization. In *Advances in Neural Information Processing Systems*, pages 5115–5124, 2017.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520. IEEE, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Leslie N Smith, Emily M Hand, and Timothy Doster. Gradual dropin of layers to train very deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4763–4771, 2016.

Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In *International Conference on Computational Learning Theory*, pages 545–560. Springer, 2005.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *ICML*, pages 1058–1066, 2013.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

Ke Zhai and Huan Wang. Adaptive dropout with rademacher complexity regularization. In *International Conference on Learning Representations*, 2018.