Bayesian estimation and model averaging of convolutional neural networks by hypernetwork

Ukai, Kenya

Matsubara, Takashi

Uehara, Kuniaki

NOLTA, IEICE

Paper

Bayesian estimation and model averaging of convolutional neural networks by hypernetwork

Kenya Ukai [1 a)], *Takashi Matsubara* [1], *and Kuniaki Uehara* [1]

[1] *Graduate School of System Informatics, Kobe University*
1-1 Rokkodai, Nada, Kobe, Hyogo 657-8501, Japan

a) *ukai@ai.cs.kobe-u.ac.jp*

Received April 11, 2018; Revised August 23, 2018; Published January 1, 2019

Abstract: Neural networks have a rich ability to learn complex representations and have achieved remarkable results in various tasks. However, they are prone to overfitting owing to the limited number of training samples and regularizing the learning process of neural networks is essential. In this paper, we propose a regularization method that estimates the parameters of a large convolutional neural network as probabilistic distributions using a hypernetwork, which generates the parameters of another network. Additionally, we perform model averaging to improve the network performance. Then, we apply the proposed method to a large model such as wide residual networks. The experimental results demonstrate that our method and its model averaging outperform the commonly used maximum a posteriori estimation with L2 regularization.

Key Words: convolutional neural network, hypernetwork, Bayesian estimation, image recognition

1. Introduction

Neural networks have a rich ability to learn complex representations and have achieved remarkable results in various tasks; they have surpassed human performance in visual recognition [1] and speech recognition [2, 3]. They have also improved the performance of machine translation [4]. However, they are prone to overfitting owing to the limited number of training samples; regularizing the learning process of neural networks is essential [5].

Many studies have addressed overfitting [5, 6]. Weight decay prevents the weights from growing excessively large. This works as a gradient descent on a quadratic weight term and hence it is referred to as L2 regularization. Dropout [7] prevents units from excessively co-adapting by randomly dropping units from a neural network during training. Although these are heuristic methods, we can interpret them as Bayesian methods; L2 regularization is equivalent to the introduction of a Gaussian prior of the parameters and the maximum a posteriori (MAP) estimation of the parameters [6], and dropout can be interpreted as the variational inference that minimizes the Kullback-Leibler divergence from an approximate distribution to the posterior of a Gaussian process [8]. Thus, the regularization methods of neural networks are closely related to the Bayesian approaches. methods [9–11]. Such models are called Bayesian neural networks (BNNs). The most popular approach is variational inference which restricts the variational posterior to a simple family of distributions. Although BNNs perform better than non-Bayesian methods in some tasks, they have only been applied to pure fully-connected networks or small convolutional neural networks (CNNs).

However, some studies have proposed a type of neural network called a hypernetwork [12]. A hypernetwork is a neural network that outputs the parameters of another neural network. Ha et al. [12] used hypernetworks to reduce parameters of large CNNs but reduced the performance drastically decreased. Krueger et al. [10] represented complex approximate posteriors using hypernetworks and applied them to BNNs. However, this method restricts the hypernetwork's structure.

Following the previous studies, we propose a new regularization method for large-scale CNNs, especially residual networks [13]. Similar to Krueger et al. [10], we use hypernetworks to express the approximate posterior of the parameters. Thanks to the estimation of the parameters as probabilistic distributions, we expect that the stochastic behavior of the parameters regularizes the learning process. Moreover, we can perform Bayesian model averaging to improve the performance. We do not employ generally used variational inference, because of the difficulty of applying it to large networks; otherwise, we would need to calculate the likelihood of a given parameter value to perform variational inference with Kullback-Leibler divergence and restrict the structure of the hypernetworks. Alternatively, we minimize the cross-entropy loss directly. We can thus build hypernetworks more freely and reduce the entire network size. We applied the proposed method to wide residual networks [14] and evaluated it on CIFAR-10 [15]. The experimental results demonstrate that the regularization of our method and its model averaging outperform the MAP estimation with L2 regularization.

2. Related works

2.1 Hypernetworks

A hypernetwork is a neural network that outputs the parameters of another neural network (we call it a "primary network"). Let $F(x; w) : X \times W \to Y$ be a primary network, where x is the input such as an image, y is the output such as a class label, and w is the parameters. A hypernetwork is defined as $G(z; \theta) : Z \times \Theta \to W$, where z is the input and θ is the parameters of the hypernetwork. Figure 1 shows the structure of the neural network using the hypernetwork. When training the primary network with hypernetworks, we obtain the optimal parameters w^* as the outputs of the hypernetworks instead of estimating w^* by minimizing the loss over the parameters w.

We can apply hypernetworks to various tasks by designing their network structures and input domains. Ha et al. [12] reduced the number of the parameters of wide residual networks (WideResNet) [14], which perform well on image classification tasks. They employed a single hypernetwork; they grouped every N parameters of the primary network and replaced each group with an output of the hypernetwork; they employed an M $(< N)$-dimensional trainable parameter as the input of the hypernetwork for each group. By sharing the hypernetwork among the groups, they reduced the number of parameters. Hypernetworks that generate multiple groups of the parameters work as relaxed weight-sharing. They regularize the learning process. Whereas they successfully reduced the number of the parameters, the experimental results showed that the classification accuracy also became worse. They suggest that sharing one hypernetwork across the network is an excessively strong assumption.

2.2 Bayesian neural networks

BNNs are neural networks whose parameters are not point estimates but posterior distributions. When training a non-Bayesian neural network, we define a loss function $\mathcal{L}(x, y, w)$, update the parameters to minimize the loss function by the gradient descent, and obtain the optimal parameters w^*. The cross-entropy $-\log p(y|x; w)$ is often used as the loss function. This scheme is the maximum likelihood estimation (MLE). We can set prior distributions of the parameters, use $-\log p(w|x, y) \propto -\log p(y|x; w)p(w)$ as the loss function, and obtain the optimal parameters w^*. This scheme is the maximum a posteriori (MAP) estimation. These estimations are point estimations of the parameters. Bayesian estimation is an estimation of the posteriors of the parameters.

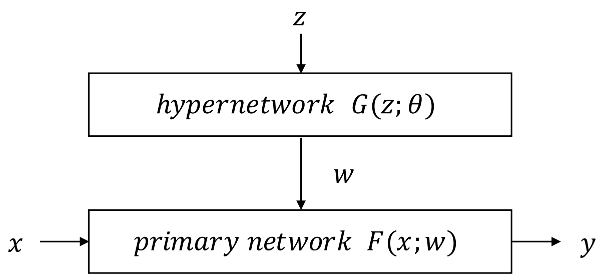


Fig. 1. Structure of neural network using hypernetwork.

We can regularize the learning process by the probabilistic behavior of the parameters and express uncertainty.

In the point estimation, the network output is also estimated as a point. When we estimate the posterior distributions, we need to integrate over the posterior to calculate the output of the network. Because the integration is very expensive, MAP estimation or Monte Carlo sampling is usually employed. Monte Carlo sampling of the posterior of the parameters is expressed by the following equation:

$$p(y|x,\mathcal{D}) = \sum_{w\in\mathcal{W}} p(y|x,w,\mathcal{D})p(w|\mathcal{D}) \tag{1}$$

This method is called Bayesian model averaging, which involves creating an implicit ensemble and which is expected to improve the performance.

Blundell et al. [9] proposed Bayes by Backprop, which estimates the posterior of the parameters of a neural network with variational inference. They used a factorial Gaussian distribution as a prior and a variational posterior; their method only captures a single mode of the true posterior, and higher-order relationship among the posteriors. Krueger et al. [10] proposed hypernetworks to represent complex approximate posteriors. They used invertible-structured neural networks called *real-NVP* [16] as hypernetworks. This enables one to calculate the exact likelihood of a given weight parameter set. Real-NVP requires repeated coupling of layers and the same dimensions of the input as those of the output (i.e., weight parameter); a simple application to a deep CNNs leads to extremely large hypernetworks and many parameters. To prevent this issue, Krueger et al. [10] employed weight normalization [17] and applied hypernetworks only to scale parameters of weight normalization instead of the whole weight parameters. The original study [17] of the weight normalization revealed that weight normalization has a limited performance (or does not work) for large-scale CNNs compared to batch normalization [18] (see also [19]). As a result, the performances of Bayesian hypernetworks are highly limited for image processing. Krueger et al. addressed the problem by using Weight Normalization [17] and only generating its scale parameters. Generally, the performance of Weight Normalization is worse than that of batch normalization [18]. Additionally, their regularization performance is limited because only a small subset of the parameters are regularized.

2.3 Convolutional neural networks

CNNs perform well on image classification tasks. In particular, a residual network (ResNet) and its variants achieved state-of-the-art results [13, 20, 21]. The structure of ResNet prevents gradient vanishing. Figure 2 shows the structure of a typical ResNet. A ResNet is composed of three parts: a preconvolution layer, sequence of residual blocks (ResBlocks), and fully connected layer. A ResBlock is composed of convolution layers, batch normalization layers, and activation functions, and their order may vary [13].

Many studies have improved the performance of residual networks. Zagoruyko et al. [14] introduced wide residual networks (WideResNet) and showed that increasing the number of the channels improves the performance. Xie et al. [22] introduced ResNeXt and improved the performance by arranging ResBlocks in parallel.

For applying hypernetworks, some variants such as ResNet, WideResNet, and ResNeXt are convenient because the structures are simple; they use many ResBlocks of the same sizes repeatedly.

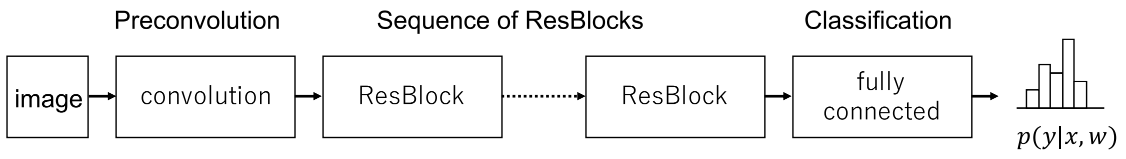


Fig. 2. Structure of a ResNet [13].

Conversely, some networks such as PyramidalResNet [21] and DenseNet [20] have a wide variety of ResBlocks and applying hypernetworks is more troublesome.

3. Proposed method

3.1 Estimating parameters as probabilistic distribution by hypernetworks

Unlike previous studies [9, 10, 12], we aim at a comparative or superior performance to the MAP estimation.

As mentioned above, in the learning of neural networks for image classification tasks, we minimize the loss function $\mathcal{L}(x, y, w)$ where x is an image, y is a true class label, and w is the parameters. On the other contrary, for a network with a hypernetwork $g(z; \theta)$, we train the hypernetwork's parameters θ instead of the primary networks' parameters w. We draw a sample z from a prior distribution $p(z)$ and input it to the hypernetwork. We used the output $g(z; \theta)$ as the primary network's parameters $w = g(z; \theta)$; the output w forms a distribution $q(w; \theta)$ implicitly [23, 24].

Ordinarily, the prior $p(w)$ and variational posterior $q(w)$ of the parameters w have been expressed as known distributions such as an isotropic Gaussian distributions [9, 25]. This assumption could excessively constrain the parameters by ignoring desired correlations between the weight parameters. Conversely, we do not assume the variational posterior as an explicit formulation of a density function. Instead, we consider the hypernetwork's output w given a random sample z as a sample from the variational posterior of the primary network's parameters w. Thanks to this assumption, the primary network's parameters $q(w)$ could have a variational posterior with a complicated shape.

Unlike previous studies [9, 10], we cannot minimize the divergence of the variational posterior from the prior directly, but we can constrain the variational posterior by introducing the weight decay to the hypernetwork's parameters θ and the prior $p(z)$ to the input z. In other words, we implicitly introduce the prior of the primary network's parameters w using the hypernetwork.

Specifically, the joint distribution of the primary network and the hypernetwork is

$$p(y, x, \theta) = \int p(y|x, w)p(w|\theta)p(\theta)p(x)\mathrm{d}w$$

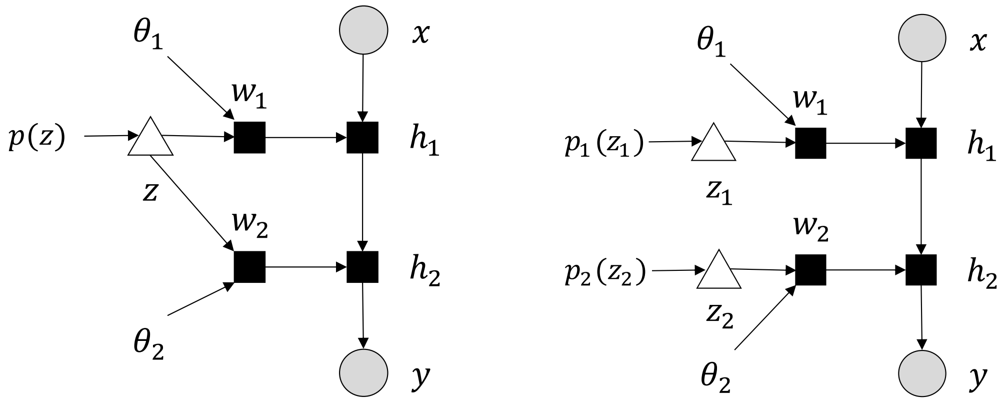
where $p(w|\theta) = \int \delta(w = g(z; \theta))p(z)\mathrm{d}z$. Here, we obtain the weight parameters w of the primary network as a sample drawn from the hypernetwork $p(w|\theta)$, such as deep implicit generative models [24]. Because $p(w|\theta)$ is not explicitly available, unlike previous studies [9, 10], we approximate it by $\hat{p}(w|\theta) = \sum_{n=1}^{N} \delta(w = g(z_n; \theta))$, where $z_{1:N} \sim p(z)$. The approximate joint distribution is

$$\hat{p}(y, x, \theta) = \sum_{n=1, z_n \sim p(z)}^{N} p(y|x, w = g(z_n; \theta)))p(\theta)p(x)$$

We introduce an isotropic Gaussian prior $p(\theta)$ to the hypernetwork's parameter θ and obtain its approximate MAP estimation via weight decay (which is equivalent to L_2-regularization). Given an input x and the hypernetwork's parameter θ, the posterior probability of the class label y is

$$\hat{p}(y|x, \theta) = \sum_{n=1, z_n \sim p(z)}^{N} p(y|x, w = g(z_n; \theta))$$

In our implementation, we use $N = 1$. Then, the loss function $\mathcal{L}$ is the cross-entropy of the true class label $p(y|x, \mathcal{D})$ obtained from the dataset $\mathcal{D}$ and the posterior $\hat{p}(y|x, \theta)$, in addition to the aforementioned L_2-regularization. The derivative of the loss function $\mathcal{L}$ with respect to the hypernetwork's parameters θ can be obtained by the backpropagation algorithm;



(a) all-in-one strategy (b) layer-wise strategy

Fig. 3. Variation of strategies for generating parameters.

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial w}\frac{\partial w}{\partial \theta}\Big|_{w=g(z;\theta)}$$

3.2 Strategy for generating parameters

Generating the parameters of the primary network constrains on the parameters. It builds the relationship between the parameters depending on the structure of the hypernetwork and its input.

We propose some strategies for generating the parameters. We call one of them *all-in-one*, which builds the relationship among all the parameters. Figure 3 (a) shows a graphical model, where a deterministic variable is denoted by a square, a sample from a probabilistic distribution with static parameters is denoted by a triangle, and an observed variable is denoted by a circle. In this strategy, all the parameters of the primary network are generated from a single sample of the distribution $p(z)$; all the parameters are related to each other.

Otherwise, we divide the parameters into N groups and builds the relationship only inside each group. For example, we can make a group for each layer. Figure 3 (b) shows the graphical model for the case of $N = 2$. In this strategy, the parameters are generated from a sample of distribution $p(z_i)$ for each layer i independently. We call the strategy *layer-wise*. Note that we can reduce the number of the parameters by sharing the hypernetwork among the groups, but we do not do this because it reduces the performance [21]. When applying the strategy to WideResNet [14], instead of dividing the parameters into each layer, we can do it into each ResBlock, which we call *block-wise*.

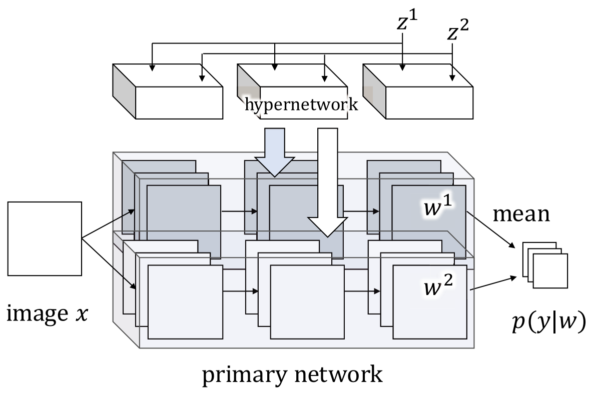
We employ a network of the same structure for each strategy. We build a hypernetwork $g(z;\theta_i)$ for each layer i of the primary network. For the *all-in-one* strategy, we sample only one point from a distribution $p(z)$ and input it to all the hypernetworks. This enables each hypernetwork to know input from the others. For the *layer-wise* and *block-wise* strategies, we independently sample a point from $p(z_i)$ for each layer or ResBlock and input them. This prevents each group from co-adapting.

3.3 Generating parameters of ResNet by hypernetworks

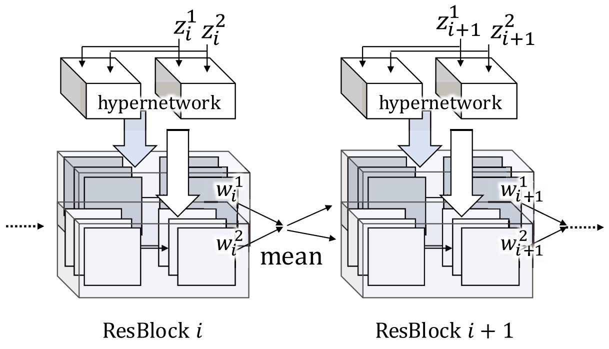
Although we mentioned that we build a hypernetwork for each layer, we isolate some layers from the hypernetworks. We do not generate the parameters of the layers other than ResBlocks: the pre-convolution layer and the last fully connected classification layer. We also exclude affine parameters of the batch normalization. Alternatively, we apply weight decay to them. We build a hypernetwork for each convolution layer of residual blocks. For convolution layer i, which has a kernel of $f_{size} \times f_{size}$, an input of N_{in} channels, and an output of N_{out} channels, we build a hypernetwork $g_i(z_i;\theta_i)$, which outputs an $f_{size} \times f_{size} \times N_{in} \times N_{out}$-dimensional weight parameter. We only generate the weight parameters of the filters, because the convolution layers have no bias term in ResNet.

3.4 Model averaging

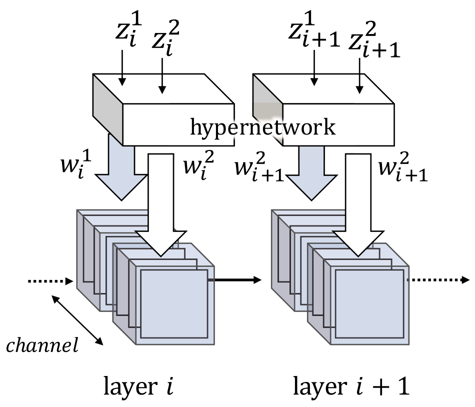
Because we estimate the variational posterior of the parameters, we can perform model averaging. If samples from the learned distribution are diverse , the model averaging improves the classification



(a) Branching



(b) Parallelizing



(c) Widening

Fig. 4. Possible methods of the model averaging for ResNet.

performance like an ensemble.

In this study, we use different ways of the model averaging depending on the strategies. For the *all-in-one* strategy, we generate N sets of the parameters and build N primary networks. We perform the model averaging by averaging the outputs of the N networks. We call this $\times N$ *branching*. Figure 4 (a) shows the diagram of $\times 2$ *branching*. z_i is a sample from a prior $p(z)$ and transformed to parameters w_i by a hypernetwork. For the *block-wise* strategy, we generate N residual blocks and put them in parallel, as in ResNeXt [22]. We call this $\times N$ *parallelizing*. Figure 4 (b) shows the diagram of $\times 2$ *parallelizing*. For the *layer-wise* strategy, we increase the number of the channels N times, as in wide residual networks. We call this $\times N$ *widening*. Figure 4 (c) shows the diagram of $\times 2$ *widening*. Note that *parallelizing* and *layer-wise* are not the ordinary form of model averaging mentioned in Sec. 2.3 because they average subsets inside the model.

3.5 Finetuning batch normalization at inference

A residual network and its variations use batch normalization [18]. Batch normalization whitens each hidden activation with its mean and variance over the mini-batch during the training and uses those of the whole training dataset at the inference. When applying our methods, the output distribution of each layer varies at the inference, owing to the stochastic behavior of the parameters. To address this problem, we recalculate the mean and variance of the batch normalization according to the parameters generated from the hypernetworks. Specifically, after generating the parameters of the primary network, we train the mean and variance of batch normalization for one epoch with the generated parameters. When performing model averaging, we duplicate the batch normalization's affine parameters because we do not generate them from the hypernetworks.

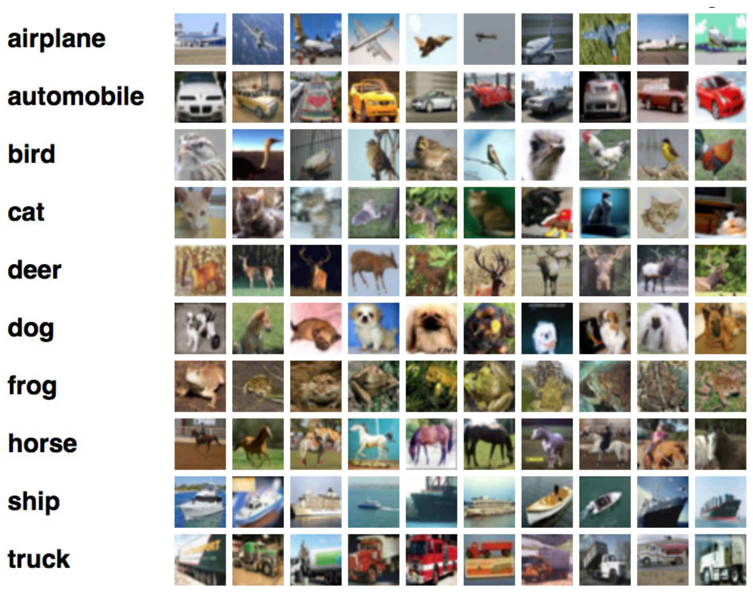


Fig. 5. Example image of CIFAR-10 [15]. Source: http://www.cs.toronto.edu/~kriz/cifar.html

4. Experiments and results

4.1 Experimental settings

We evaluated our methods on the CIFAR-10 [15] dataset. It consists of 50k training samples and 10k testing samples. Each sample is a 32×32 color natural image in one of 10 classes. Figure 5 shows example images. In accordance with the experiments of Zagoruyko et al. [14], we applied mean/std normalization, random horizontal flipping, random cropping from images padded by four pixels on each side. These normalization and augmentation processes are commonly used for classifications of CIFAR-10 [14].

First, we applied the proposed method to WideResNet [14]. WideResNet is a very deep CNN based on residual networks [13]. We used the same architectures as the original study [14]. At the time of model averaging, we inserted the averaging layer to match the dimensions: at the output of each ResBlock for *parallelizing*, at the output of the last convolutional layer for *widening*, and at the output of the last fully connected layer for *branching* (see Fig. 4).

For the hypernetwork architecture, we used neural networks of one hidden layer with the hidden units h_{dim} and the input units z_{dim} for each convolutional layer. We used $z_{dim} = h_{dim} = 32$ or $z_{dim} = h_{dim} = 16$ according to the limitation of computational resources. From the results of the preliminary experiments, to prevent vanishing of the random input of the hypernetworks, we did not use a bias term in the hypernetworks. We used the ReLU as the activation function.

We followed the training methods in the original study of WideResNet [14] unless otherwise stated. We trained all the parameters by stochastic gradient descendent (SGD) with Nesterov momentum and cross-entropy loss, the momentum parameter of 0.9, and mini-batch size of 128. The learning rate was set to 0.1 and dropped at 40%, 60%, and 80% of all the training epochs by 0.2. Although weight decay was applied to all the parameters in the original study [14], we did not apply it to the parameters generated by the hypernetworks. Instead, we applied a weight decay factor of 0.0005 to the other parameters. We initialized the parameters except for the weights of the output layers of the hypernetworks, by He's method [1]. For the weights of the output layers, to stabilize the training, we sampled from uniform distributions with ranges 1/10 of those of He's method [1].

For the evaluation, the accuracies can vary owing to the stochastic behavior of the training methods. We trained the networks three times with different initial values and evaluated them by the median accuracies. Medians are commonly used to evaluate the accuracy of classification models such as residual networks [13]. For the networks using the proposed method, because random vectors are used as input of the hypernetworks, we need to take account of the stochasticity in the evaluation

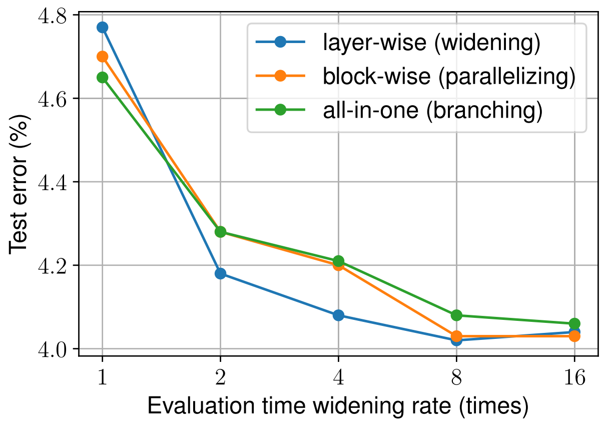


Fig. 6. Test error rates on CIFAR-10 for WideResNet28-4 generated by the hypernetwork with the prior of $\mathcal{N}(0,1)$ with strategies.

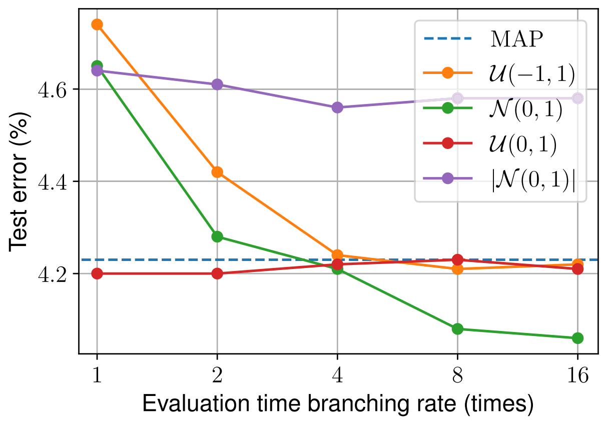


Fig. 7. Test error rates on CIFAR-10 for WideResNet28-4 generated by the hypernetwork with the priors of different distributions.

time. We ran the trained networks five times with different random vectors and evaluated them by the median accuracies.

4.2 Hyperparameters

We expected that the regularization effects depend on the strategy for generating the parameters. We compared the test accuracy among three types of strategies; *all-in-one*, *layer-wise*, and *block-wise*. Figure 6 shows the test accuracy and the model averaging rate of each strategy. We performed *branching* for *all-in-one*, *parallelizing* for *layer-wise*, and *widening* for *block-wise*. We used $N(0,1)$ as a prior $p(z)$ and trained the networks for 800 epochs.

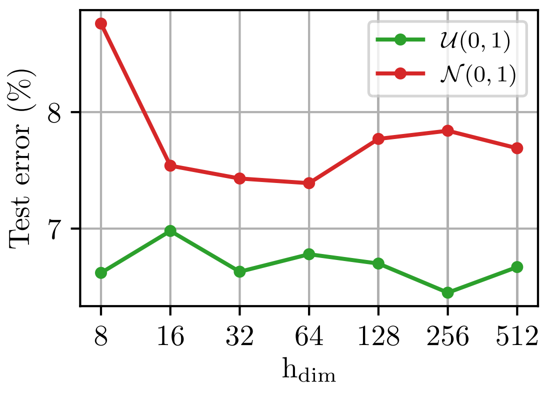
We also examined four types of prior distributions $p(z)$ and compared them: $\mathcal{N}(0,1)$, $\mathcal{U}(-1,1)$, $\mathcal{U}(0,1)$, and $|\mathcal{N}(0,1)|$. The first two distributions take positive and negative values and the others take only positive values. Figure 7 shows the test error rates on CIFAR-10 for WideResNet28-4 with different priors. We used the *all-in-one* strategy.

We also examined the relationship between the size of the hypernetworks and the generalization performance. Figure 8 shows the test error rates on CIFAR-10 for WideResNet28-4 and WideResNet28-1 with different hypernetwork unit sizes $h_{size} = z_{size}$. We used the *all-in-one* strategy and $\mathcal{U}(0,1)$ or $\mathcal{N}(0,1)$ as a prior $p(z)$.

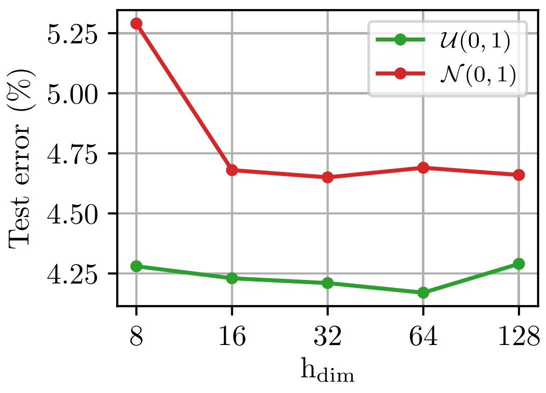
4.3 Comparison with other CNNs and other datasets

We evaluated the regularization of the proposed method and the effect of model averaging, by comparing the test accuracy with MAP estimation. For the proposed method, we employed the *all-in-one* hypernetwork with the prior $p(z)$ of $\mathcal{U}(0,1)$.

We applied the proposed hypernetwork to WideResNet, ResNeXt and Pyramidal ResNet on CIFAR-



(a) WideResNet28-1



(b) WideResNet28-4

Fig. 8. Test error rates on CIFAR-10 with different hypernetwork unit sizes $h_{dim}(= z_{dim})$.

Table I. Test error rates on CIFAR-10 for various CNNs.

	WideResNet28-10		ResNeXt-29-8-64d		Pyramidal ResNet-110-48	
Methods	×1	×16	×1	×16	×1	×16
MAP	3.90%	—	4.03%	—	4.59%	—
hypernetwork	**3.76%**	**3.73%**	**3.92%**	**3.91%**	4.64%	4.61%

Table II. Test error rates on SVHN for WideResNet16-4.

Methods	prior $p(z)$	×1	×16
MAP	—	1.90%	—
hypernetwork	$\mathcal{U}(0,1)$	1.93%	1.93%
	$\mathcal{N}(0,1)$	1.90%	**1.80%**

Table III. Test error rates on ImageNet for ResNet50 with the prior of $U(0,1)$.

	Top 1		Top 5	
Methods	×1	×16	×1	×16
MAP	23.84%	—	7.06%	—
hypernetwork	25.97%	25.87%	8.17%	8.16%

10. The results are summarized in Table I. As WideResNet, we used *WideResNet28-10*. As ResNeXt, we used *ResNeXt29-8-64d*, consisting of 29 convolution layers with a cardinality of 8 and base channel widths of 64. As Pyramidal ResNet, we used *Pyramidal ResNet 110-48*, consisting of 110 convolution layers with a widening factor of $\alpha = 48$. The hypernetworks' unit size z were 16 for WideResNet and ResNeXt, and 32 for Pyramidal ResNet. The hyperparameters were the same as in the original studies, except for the training epochs. We trained WideResNet for 800 epochs, ResNeXt for 900 epochs and Pyramidal ResNet for 600 epochs.

We also evaluated our proposed method on SVHN [26] by applying it to the network architecture called WideResNet16-4 [14]. The hyper-parameters were the same as in [14]. We employed the *all-in-one* hypernetwork with the prior $p(z)$ of $\mathcal{U}(0,1)$ and $\mathcal{N}(0,1)$. The hypernetworks unit size z_{dim} was 32. The results are summarized in Table II.

We also evaluated proposed method on ImageNet [26]. We used the network architecture called ResNet50 [13]. The hyperparameters were the same as in the original studies. We employed the *all-in-one* hypernetwork with the prior $p(z)$ of $\mathcal{U}(0,1)$. The hypernetworks unit size z_{dim} was 32. The results are summarized in Table III. The result is based on one run, owing to the limitation of

Table IV. Test error rates on CIFAR-10 for WideResNet28-4 with different regularization methods.

		number of models	
estimation method	prior $p(z)$	$\times 1$	$\times 16$
Ensemble-MLE	—	6.05%	4.75%
Ensemble-MAP	—	4.23%	3.14%
Bayesian-Hypernetworks	—	failed	failed
Hypernetwork	$\mathcal{N}(0, 1)$	4.65%	4.06%

computational resources.

4.4 Comparison with other methods

We also evaluated the basic ensemble of the models trained with MLE and MAP estimation and compared them with our methods. We call them *Ensemble-MLE* and *Ensemble-MAP*. We used the models trained with different sets of initial weight parameters. The results of *Ensemble-MLE* and *Ensemble-MAP* are based on one run. We used WideResNet28-4 and trained the networks for 800 epochs.

We also tried to examine the works by Krueger et al. [10]. Because they employed weight normalization, we examined it and confirmed that it does not work well for WideResNet.

The results are summarized in Table IV.

5. Discussion

5.1 Strategy for generating parameters

In this section, we discuss the strategy. Following Fig. 6, the accuracy was improved in all strategies. There was no obvious difference between them at $\times 16$, but at $\times 2, 4$, and 8, *widening* achieved the highest improvement. This is because the number of parameters increases when performing model averaging. For *branching* and *parallelizing*, when performing $\times N$ model averaging , the number of parameters also increases N times. For *widening*, it increases N^2 times. When we take this point into consideration, there is no obvious difference among the strategies. Recall that widening and parallelizing are not the ordinary model averaging and are not guaranteed to work well. However, a previous study [27] showed that the ResNet behaves like ensembles of relatively shallow structures and therefore, all the strategies work well. Although we used the network of the same structure for each strategy in the experiment, we could use different network structures to improve these results. For example, we could use a single large hypernetwork that outputs all the parameters for the *all-in-one* strategy in order to build the relationship among all the parameters. However, it makes the hypernetwork so large that we cannot use large networks such as WideResNet as the primary network.

5.2 Prior $p(z)$

Here, we examine the priors. In Fig. 7, there was a difference between the priors $p(z)$. The accuracies of our models with $\mathcal{U}(-1, 1)$ and $\mathcal{N}(0, 1)$ are worse than that of the MAP estimation at $\times 1$ but are better at $\times 16$. In contrast, the accuracies of our models with $\mathcal{U}(0, 1)$ and $|\mathcal{N}(0, 1)|$ are competitive with that of the MAP estimation at $\times 1$ but show limited improvement at $\times 16$. For ensembles, the improvement of the accuracy generally depends on the diversity of the models ensembled [28]. Because model averaging is deeply related to the ensemble, we can evaluate model averaging by the evaluation methods used for the ensemble. Although there is no absolute indicator for the diversity that prompts the improvement of accuracy, we can evaluate the diversity by the double fault [29]. The double fault is defined as the ratio of test data that two models misclassify together. A small number of samples misclassified only once suggests that the two models classify images with similar boundaries and the diversity is low.

To determine the values more accurately, we used four models (it may thus be better to call it quadruple fault). When a model is trained from scratch four times and each trial converges to the

Table V. Quadruple faults ($\times 10^4$) on CIFAR-10 for WideResNet28-4 generated by hypernetwork with different priors of distributions and vanilla WideResNet with MAP estimation.

		misclassified samples			
estimation method	prior $p(z)$	four times	three times	two times	one time
Hypernetwork	$\mathcal{N}(0,1)$	269	109	130	225
	$\mathcal{U}(-1,1)$	259	126	133	210
	$\mathcal{U}(0,1)$	389	19	20	24
	$\lvert\mathcal{N}(0,1)\rvert$	405	24	19	25
MAP	—	196	123	173	314

same parameter values, all of the four trained models classify a sample into the same class. In this case, no samples are misclassified one, two, or three times, but some samples are misclassified four times. If each trial converges to its own parameter values, each trained model classifies a sample its own measure. Then, many samples are misclassified by only one model and a limited number of samples are misclassified by all four of the trained models. Hence, many samples misclassified one to three times indicates a high diversity of the trained models, whereas a large number of samples misclassified four times indicates a limited diversity of the trained models.

For our proposed method, we generated four sets of the primary networks' parameters by a single hypernetwork and calculated the quadruple fault among networks using these parameters. For the MAP estimation, we trained four models from different initial values of the parameters and calculated the quadruple fault among them. Table V shows the results of the quadruple faults for WideResNet28-4 generated by a hypernetwork with different distributions and trained with the MAP estimation. We used the all-in-one strategy. The numbers of samples misclassified one, two, three, and four times are shown in the table. For example, the number in the column of "three times" is the number of test samples misclassified by three of the four models. Looking at the column of "four times", our models with $\mathcal{U}(0,1)$ and $|\mathcal{N}(0,1)|$ have large values and our models with $\mathcal{U}(-1,1)$ and $\mathcal{N}(0,1)$ have lower values. The MAP estimation has the lowest value. This implies that the diversities of our model with $\mathcal{U}(0,1)$ and $|\mathcal{N}(0,1)|$ are smaller than that of others because samples misclassified four times have little chance to be classified successfully when doing model averaging or ensembling. These results are consistent with the performance improvement rates of accuracy in Fig. 7.

5.3 Size of hypernetworks and performance

Here, we discuss the performance and the computational cost of our method. The additional computational cost of our method is due to the hypernetworks and the cost depends on the unit size of the hypernetworks. Figure 8 shows that the hypernetwork produced results robustly over the range of $8 < h_{dim} < 128$ for both WideResNet28-4 and WideResNet28-1. This implies that the size $z_{dim} = 16$ is sufficient for our purpose and the best size is robust to the size of the primary network.

In this case, the increase in the computational complexity is almost negligible compared to that of the primary CNN. We calculated the computational complexity following [30]; a WideResNet 28-4 requires 0.85×10^3 operations per 32×32 RGB image, and our proposed hypernetwork with $z_{dim} = 32$ increases the computational complexity by 0.18×10^3 operations per mini-batch. In the training phase, we drew a single weight set per mini-batch from a hypernetwork and used the weight set for all images in the mini-batch; this is the same strategy as [10]. Because the mini-batch size was 128 in our experiments, the increase in the computational complexity was 0.2%. With an increase in the number of images per mini-batch, the computational complexity of the hypernetwork becomes more negligible. Because we use a hypernetwork for each convolution layer, we could run the hypernetworks in parallel. However, the parallel computation of the CNN has been highly optimized by the NVIDIA cuDNN library compared to that of the fully connected networks used in the hypernetworks; a WideResNet 28-4 with our proposed hypernetwork requires a computational time approximately twice as long as the plain WideResNet. Once the library is optimized for the hypernetworks, we consider the time complexity approaches the theoretical computational complexity. In the inference phase, if we do not perform model averaging, we can fix the drawn weight set and remove the hypernetworks. Then, the computational and time complexity are identical to those of the plain CNN. The model averaging increases in the computational complexity proportionally to the number of models involved; we can adjust the trade-off between the computational complexity and the performance by determining the number of models.

5.4 Comparison with other CNNs and other datasets

In Table I, we found that the proposed method achieved better results on WideResNet and ResNeXt, and competitive results on Pyramidal ResNet. This shows the successful regularization effect of our method. ResNeXt has a similar structure to WideResNet; it has multiple ResBlocks of the same channel size. Conversely, Pyramidal ResNet has ResBlocks of various channel sizes. Therefore, our proposed method could prefer a repetitive structure rather than Pyramidal ResNet.

Following the results in Table II, the proposed method with the prior $p(z)$ of $\mathcal{U}(0,1)$ and $\mathcal{N}(0,1)$ was competitive to the baseline of SVHN [26]. With model averaging, our proposed method with the prior $p(z)$ of $\mathcal{N}(0,1)$ outperformed the baseline. These results indicate that the proposed method is applicable not only to CIFAR10 dataset but also to SVHN dataset.

Following the results in Table III, the proposed method yielded slightly worse results on ImageNet than the baseline. In the experiments on the the CIFAR-10 and the SVHN datasets, the training error converged to almost zero while the test error showed certain positive values; these results imply a slight overfitting of the CNNs. Conversely, in the experiments on ImageNet, the training error remained at a similar level to the test error. This suggests that the ResNet50 underfitted to the training set owing to its limited expression ability compared to the dataset size, and hence, it does not require regularization. Huang et al. [31] obtained a similar result. A larger network will be explored in future work.

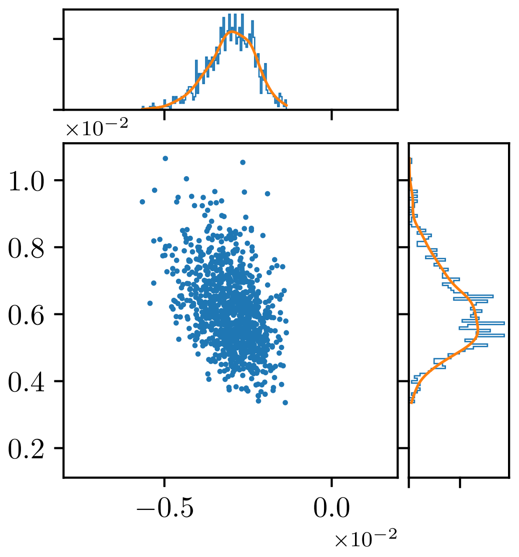
5.5 Comparison with other methods

Compared to *Ensemble-MAP*, *Ensemble-MAP* outperformed our proposed methods. Although *Ensemble-MAP* showed better results, the training cost of *Ensemble-MAP* is much larger than the proposed method; $\times 16$ *Ensemble-MAP* requires 16 times the computational cost in the training phase. However, as discussed in Sec. 5.3, our proposed model is trained only once, and the additional computational cost is negligible. Despite this fact, our proposed model can improve the performance by forming an ensemble. The improvement of diversity in the generated primary CNNs will be explored in future work.

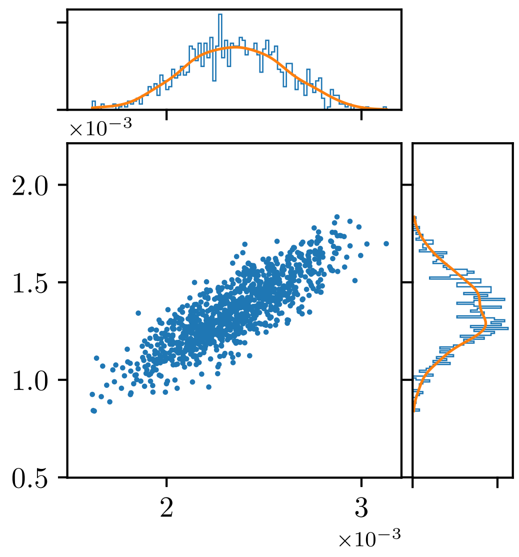
5.6 Weight distributions

We examined the posterior distribution of weights learned by our proposed hypernetworks. Figure 9 depicts 1000 scatter samples of a specific pair of weight parameters from the posterior of WideResNet28-4 with the prior $\mathcal{U}(0,1)$ and $\mathcal{N}(0,1)$. The results show that the pair of weight parameters forms distributions with covariances. With the prior of $\mathcal{N}(0,1)$, the correlations were relatively small (approximately 0.15 on average), but we can find higher-order correlations and skewness in Fig. 9 (a). In contrast, the weight parameters learned with Bayes by Backprop [9] have no covariance by definition and the MAP estimation is a point estimate. We conclude that our proposed method successfully obtains a variational posterior that has a complicated shape compared to conventional methods.

We also calculated the Pearson correlation coefficients between random pairs of weight parameters and examined the differences among the *all-in-one*, *block-wise*, and *layer-wise* strategies with the prior $\mathcal{U}(0,1)$. When we randomly chose 1000 pairs of weight parameters from the same ResBlock, the average of the absolute correlation was 0.979 for the *all-in-one* strategy, 0.970 for the *block-wise* strategy, and 0.364 for the *layer-wise* strategy. When we chose 1000 pairs of weight parameters randomly from different ResBlocks, the average was 0.943 for the *all-in-one* strategy, 0.007 for the *block-wise* strategy, and 0.008 for the *layer-wise* strategy. These results indicate that our proposed



(a) *all-in-one*, $\mathcal{N}(0,1)$



(b) *all-in-one*, $\mathcal{U}(0,1)$

Fig. 9. Scatter plot and histogram of samples from an approximate posterior of WideResNet28-4.

method obtains stochastic weight parameters with correlations, and we can restrict the co-adaptations across subparts by choosing the strategies appropriately. The error rates were nonetheless robust to the strategies as shown in Fig. 6.

6. Conclusions

We proposed new regularization methods for large-scale CNNs. We used hypernetworks as approximate posterior of the parameters and trained the model with cross-entropy loss. By estimating the parameters as probabilistic distributions, it is expected that the stochastic behavior of the parameters regularizes the learning process. Moreover, we can perform Bayesian model averaging to improve the performance. We applied the proposed method to various CNNs and demonstrated its regularization. We compared three strategies for generating the parameters and found that the regularization and the model averaging effect are similar for WideResNet. We also compared four priors $p(z)$ and found that the regularization and the model averaging effect depend on them.

Acknowledgments

This study is supported by the JSPS KAKENHI (16K12487) and the MIC/SCOPE #172107101.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *Proc. IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

[2] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "The microsoft 2016 conversational speech recognition system," *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5255–5259, 2017.

[3] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Achieving human parity in conversational speech recognition," *arXiv preprint 1610.05256*, 2016.

[4] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint 1609.08144*, 2016.

[5] K.P. Murphy, *Machine learning - a probabilistic perspective.* Adaptive computation and machine learning series, MIT Press, 2012.

[6] C.M. Bishop, *Pattern recognition and machine learning, 5th Edition.* Information science and statistics, Springer, 2007.

[7] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[8] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *Proc. 33rd International Conference on Machine Learning*, pp. 1050–1059, 2016.

[9] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," *Proc. 32nd International Conference on Machine Learning*, pp. 1613–1622, 2015.

[10] D. Krueger, C. Huang, R. Islam, R. Turner, A. Lacoste, and A.C. Courville, "Bayesian hypernetworks," *arXiv preprint 1710.04759*, 2017.

[11] N. Pawlowski, M. Rajchl, and B. Glocker, "Implicit weight uncertainty in neural networks," *arXiv preprint 1711.01297*, 2017.

[12] D. Ha, A.M. Dai, and Q.V. Le, "Hypernetworks," *arXiv preprint 1609.09106*, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *Proc. 14th European Conference on Computer Vision*, pp. 630–645, 2016.

[14] S. Zagoruyko and N. Komodakis, "Wide residual networks," *Proc. British Machine Vision Conference* (E.R.H. Richard C. Wilson and W.A.P. Smith, eds.), pp. 87.1–87.12, 2016.

[15] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images, master's thesis, dept. of comp. sci., university of toronto," 2009.

[16] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," *arXiv preprint 1605.08803*, 2016.

[17] T. Salimans and D.P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," *Advances in Neural Information Processing Systems 29*, p. 901, 2016.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proc. 32nd International Conference on Machine Learning*, pp. 448–456, 2015.

[19] I. Gitman and B. Ginsburg, "Comparison of batch normalization and weight normalization algorithms for the large-scale image classification," *arXiv*, 2017.

[20] G. Huang, Z. Liu, L. van der Maaten, and K.Q. Weinberger, "Densely connected convolutional networks," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2261–2269, 2017.

[21] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6307–6315, 2017.

[22] S. Xie, R.B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5987–5995, 2017.

[23] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A.C. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems 27*, pp. 2672–2680, 2014.

[24] D. Tran, R. Ranganath, and D.M. Blei, "Hierarchical implicit models and likelihood-free variational inference," *Advances in Neural Information Processing Systems 30*, pp. 5529–5539, 2017.

[25] D.P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint 1312.6114*, 2013.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, 2015.

[27] A. Veit, M.J. Wilber, and S.J. Belongie, "Residual networks behave like ensembles of relatively

shallow networks," *Advances in Neural Information Processing Systems 29*, pp. 550–558, 2016.

[28] Z.-H. Zhou, *Ensemble methods: foundations and algorithms.* CRC press, 2012.

[29] G. Giacinto and F. Roli, "Design of effective neural network ensembles for image classification purposes," *Image Vision Comput.*, vol. 19, no. 9-10, pp. 699–707, 2001.

[30] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *arXiv*, 2016.

[31] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K.Q. Weinberger, "Deep networks with stochastic depth," *Proc. 14th European Conference on Computer Vision*, pp. 646–661, 2016.