DEEP BAYESIAN STRUCTURE NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Bayesian neural networks (BNNs) introduce uncertainty estimation to deep networks by performing Bayesian inference on network weights. However, such models bring the challenges of inference, and further BNNs with weight uncertainty rarely achieve superior performance to standard models. In this paper, we investigate a new line of Bayesian deep learning by performing Bayesian reasoning on the structure of deep neural networks. Drawing inspiration from the neural architecture search, we define the network structure as random weights on the redundant operations between computational nodes, and apply stochastic variational inference techniques to learn the structure distributions of networks. Empirically, the proposed method substantially surpasses the advanced deep neural networks across a range of classification and segmentation tasks. More importantly, our approach also preserves benefits of Bayesian principles, producing improved uncertainty estimation than the strong baselines including MC dropout and variational BNNs algorithms (e.g. noisy EK-FAC).

1 INTRODUCTION

Bayesian deep learning aims at equipping the flexible and expressive deep neural networks with appropriate uncertainty quantification (MacKay, 1992; Neal, 1995; Hinton & Van Camp, 1993; Graves, 2011; Blundell et al., 2015; Gal & Ghahramani, 2016). Traditionally, Bayesian neural networks (BNNs) introduce uncertainty in the network weights, addressing the over-fitting issue which standard neural networks (NNs) are prone to. Besides, the predictive uncertainty derived from the weight uncertainty is also of central importance in practical applications, e.g., medical analysis, automatic driving, and financial tasks.

Modeling the uncertainty via distributions on network weights is plausible and well-evaluated (Blundell et al., 2015; Gal & Ghahramani, 2016; Ghosh et al., 2018). However, BNNs with weight uncertainty usually achieve compromising performance and hence are impractical in real-world applications (Osawa et al., 2019), due to various reasons. On one hand, specifying a sensible prior for networks weights is difficult (Sun et al., 2019; Pearce et al., 2019). On the other hand, the flexible variational posterior of BNNs comes with the inference challenges (Louizos & Welling, 2017; Zhang et al., 2018; Shi et al., 2018). Recently, the efficient particle-based variational methods (Liu & Wang, 2016) have been developed with promise, but they still suffer from the particle collapsing and degrading issues for BNNs due to the high dimension of the weights and the over-parameterization nature of such models (Zhuo et al., 2018; Wang et al., 2019).

In this work, we attempt to investigate a new direction of Bayesian deep learning that performs Bayesian reasoning on the structure of neural networks while keeping the weights as point estimates. We propose a framework, named *Deep Bayesian Structure Networks* (DBSN). Specifically, in the spirit of differentiable neural architecture search (NAS) (Liu et al., 2019; Xie et al., 2019), DBSN builds a deep network by repeatedly stacking a computational cell in which any two latent representations are connected by redundant transformations (see Figure 1). Then, we define the network structure as the random weights of these transformations[1], whose distributions are much easier to capture than those of the network weights. To jointly optimize the network weights and the parameterized distribution of the network structure, we adopt a stochastic variational inference paradigm (Blundell et al., 2015) and use the reparameterization trick (Kingma & Welling, 2013).

[1]The network structure defined in DBSN is more sensible and scalable than that considered by Dikov & Bayer (2019), i.e., the layer size and network depth, for deep neural networks.

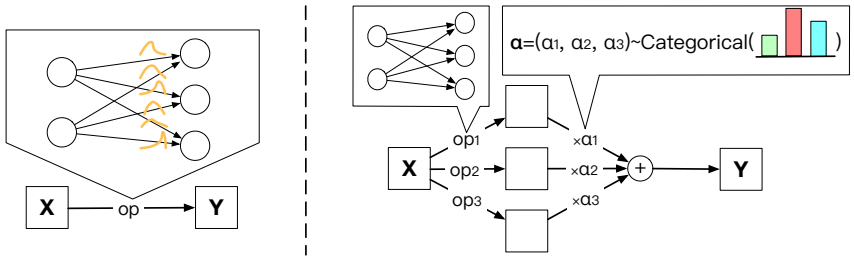

Figure 1: Left: Bayesian Neural Network with uncertainty on the weights. Right: DBSN with uncertainty on the network structure (we only depict three operations, i.e., transformations, between tensor X and tensor Y for simplicity). Note that we relax the categorical distribution with adaptive concrete distribution for back-propagation in the training.

One technical challenge is driving DBSN to achieve satisfying convergence, since the network weights can hardly fit all the structures sampled from the structure distribution. To overcome this challenge, we propose two techniques. First, we advocate reducing the variance of the sampled structures with a simple modification of the sampling procedure. Second, we suggest using a more compact structure learning space than that of NAS, to make the training more feasible and more efficient.

There are at least two motivations that make DBSN an appealing choice: 1) adjusting network structures according to the data distribution benefits the fitting of models significantly, as proved in NAS (Zoph & Le, 2016; Pham et al., 2018; Liu et al., 2019), thus DBSN shall have higher practical value than the classic BNNs; 2) DBSN places distributions on the network structures, introducing more global randomness, thus is probable to yield more diverse predictions, and ensembling them brings more calibrated uncertainty. Moreover, in the perspective of NAS, DBSN is also promising as it provides another principled way to learn network structures by substituting the Bayesian formalism for the widely used meta-learning formalism as in differentiable NAS. To empirically validate these hypotheses, we evaluate DBSN with extensive experiments. We first testify its ability of data fitting and structure learning on the challenging classification and segmentation tasks. Then, we empirically compare the quality of predictive uncertainty estimates via calibration, which is a common concern in the community. We further evaluate the predictive uncertainty on adversarial examples and out-of-distribution samples, drawn from different distributions from the training data, to verify whether the model *knows what it knows*. At last, we perform an experiment to validate the potential usage of DBSN in the one-shot NAS (Bender et al., 2018; Guo et al., 2019). Surprisingly, across all the tasks, DBSN consistently achieves comparable or even better results than the strong baselines.

2 Background

We first review the necessary background for DBSN and then elaborate DBSN in the next section.

2.1 Stochastic Variational Inference for BNNs

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ be a set of N data points. Bayesian neural networks (BNNs) are typically defined by placing a prior $p(\boldsymbol{v})$ on some variables of interest (e.g., the network weights or the network structure) and the likelihood is $p(\mathcal{D}|\boldsymbol{v})$. Directly inferring the posterior distribution $p(\boldsymbol{v}|\mathcal{D})$ is intractable because it is hard to integrate w.r.t. $\boldsymbol{v}$ exactly. Instead, variational BNNs (Hinton & Van Camp, 1993; Graves, 2011; Blundell et al., 2015) suggest approximating $p(\boldsymbol{v}|\mathcal{D})$ with a $\boldsymbol{\theta}$-parameterized distribution $q(\boldsymbol{v}|\boldsymbol{\theta})$ by minimizing the Kullback-Leibler (KL) divergence between them:

$$\min_{\boldsymbol{\theta}} D_{\mathrm{KL}}(q(\boldsymbol{v}|\boldsymbol{\theta})\|p(\boldsymbol{v}|\mathcal{D})) = -\mathbb{E}_q[\log p(\mathcal{D}|\boldsymbol{v})] + D_{\mathrm{KL}}(q(\boldsymbol{v}|\boldsymbol{\theta})\|p(\boldsymbol{v})). \tag{1}$$

To solve this, the most commonly used method is the low-variance reparameterization trick (Kingma & Welling, 2013), which replaces the sampling procedure $\boldsymbol{v} \sim q(\boldsymbol{v}|\boldsymbol{\theta})$ with the corresponding deterministic operation, to enable the direct gradient back-propagation through $\boldsymbol{\theta}$.

2.2 CELL-BASED DIFFERENTIABLE NEURAL ARCHITECTURE SEARCH

Cell-based NAS has shown promise (Zoph et al., 2018; Pham et al., 2018) and been developed to be differentiable for better scalability (Liu et al., 2019; Xie et al., 2019; Weng et al., 2019). Generally, the network in cell-based differentiable NAS[2] is composed of a sequence of cells which have the same internal structure and are separated by upsampling or downsampling modules. Every cell contains B sequential nodes (i.e., tensors): $\boldsymbol{N}^1, \ldots, \boldsymbol{N}^B$. There are K redundant operations $o_1^{(i,j)}, \ldots, o_K^{(i,j)}$ from $\boldsymbol{N}^i$ to $\boldsymbol{N}^j$ so long as $i < j$. Let $\boldsymbol{w}$ denote the network weights. The network structure is defined as $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}^{(i,j)} | 1 \leq i < j \leq K\}$ where $\boldsymbol{\alpha}^{(i,j)} \in \Delta^{K-1}$ corresponds to the weights of available operations from $\boldsymbol{N}^i$ to $\boldsymbol{N}^j$. Therefore, the computation between the two nodes can be calculated by

$$\boldsymbol{N}^{(i,j)} = \sum_{k=1}^{K} \boldsymbol{\alpha}_k^{(i,j)} \cdot o_k^{(i,j)}(\boldsymbol{N}^i; \boldsymbol{w}). \tag{2}$$

Then, the node $\boldsymbol{N}^j$ is calculated by summing all the information from its predecessors:

$$\boldsymbol{N}^j = \sum_{i<j} \boldsymbol{N}^{(i,j)}. \tag{3}$$

Meta-learning-like gradient descent is adopted for optimization to reduce the prohibitive computational cost needed by RL or evolution (Liu et al., 2019; Xie et al., 2019). However, the goal of the optimization is the network structure instead of the model performance. Thus, after training, this kind of NAS needs to prune the searched structure and re-train a new network model with the compact structure for performance comparison, which is labor-intensive and is avoided in our work.

3 DEEP BAYESIAN STRUCTURE NETWORKS

In this work, we propose a Bayesian structure learning framework for the deep neural networks, to obtain expressive models with well-calibrated uncertainty estimation. Concretely, we follow the network design of NAS but we view $\boldsymbol{\alpha}$ as Bayesian variables and $\boldsymbol{w}$ as point estimates. Assuming that the approximate posterior as well as the prior of $\boldsymbol{\alpha}$ are fully factorizable and every $\boldsymbol{\alpha}_{(i,j)}$ follows a categorical distribution, we have: $p(\boldsymbol{\alpha}) = \prod_{i<j} p(\boldsymbol{\alpha}^{(i,j)})$, $q(\boldsymbol{\alpha}|\boldsymbol{\theta}) = \prod_{i<j} q(\boldsymbol{\alpha}^{(i,j)}|\boldsymbol{\theta}^{(i,j)})$, where $\boldsymbol{\theta} = \{\boldsymbol{\theta}^{(i,j)} | 1 \leq i < j \leq K\}$ and $\boldsymbol{\theta}^{(i,j)} \in \mathbb{R}^K$ represents the trainable categorical logits. Then we rewrite the approximation error:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{w}) = -\mathbb{E}_{q(\boldsymbol{\alpha}|\boldsymbol{\theta})}[\log p(\mathcal{D}|\boldsymbol{\alpha}, \boldsymbol{w})] + D_{\mathrm{KL}}(q(\boldsymbol{\alpha}|\boldsymbol{\theta}) \| p(\boldsymbol{\alpha})). \tag{4}$$

Notably, minimizing $\mathcal{L}$ w.r.t. $\boldsymbol{\theta}$ and $\boldsymbol{w}$ relates to Bayesian reasoning and the maximum likelihood estimation, respectively. Thus, the optimization of the network structure and weights can be unified as $\min_{\boldsymbol{\theta}, \boldsymbol{w}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{w})$. To resolve this, we relax both $p(\boldsymbol{\alpha}^{(i,j)})$ and $q(\boldsymbol{\alpha}^{(i,j)}|\boldsymbol{\theta}^{(i,j)})$ to be the concrete distributions (Maddison et al., 2016) which yield samples via the softmax transformation:

$$\boldsymbol{\alpha} = g(\boldsymbol{\theta}, \boldsymbol{\epsilon}) = \{\mathrm{softmax}((\boldsymbol{\theta}^{(i,j)} + \boldsymbol{\epsilon}^{(i,j)})/\tau)\}, \tag{5}$$

where $\boldsymbol{\epsilon} = \{\boldsymbol{\epsilon}^{(i,j)} \in \mathbb{R}^K | \boldsymbol{\epsilon}_k^{(i,j)} \sim \text{Gumbel i.i.d.}\}$ are the Gumbel variables and $\tau \in \mathbb{R}_+$ is the temperature. Then we derive the following gradient estimators:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{\epsilon}}[-\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}|g(\boldsymbol{\theta}, \boldsymbol{\epsilon}), \boldsymbol{w}) + \nabla_{\boldsymbol{\theta}} \log q(g(\boldsymbol{\theta}, \boldsymbol{\epsilon})|\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \log p(g(\boldsymbol{\theta}, \boldsymbol{\epsilon}))], \tag{6}$$

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{\epsilon}}[-\nabla_{\boldsymbol{w}} \log p(\mathcal{D}|g(\boldsymbol{\theta}, \boldsymbol{\epsilon}), \boldsymbol{w})]. \tag{7}$$

The first term in Eq. (6) corresponds to the gradient of the negative log likelihood and we leave the estimation of the last two terms (log density) in the next section. In practice, we approximate the expectation in Eq. (6) and Eq. (7) with T Monte Carlo (MC) samples, and update the structure and the weights simultaneously.

After training, we gain the following predictive distribution:

$$p(y|x_{new}, \boldsymbol{w}^*) = \mathbb{E}_{q(\boldsymbol{\alpha}|\boldsymbol{\theta}^*)}[p(y|x_{new}, \boldsymbol{\alpha}, \boldsymbol{w}^*)], \tag{8}$$

where $\boldsymbol{\theta}^*$ and $\boldsymbol{w}^*$ denote the converged parameters. Eq. (8) implies that the model predicts by ensembling the predictions of the networks whose structures are randomly sampled.

[2]We will refer to the cell-based differentiable NAS as NAS for short if there is no misleading.

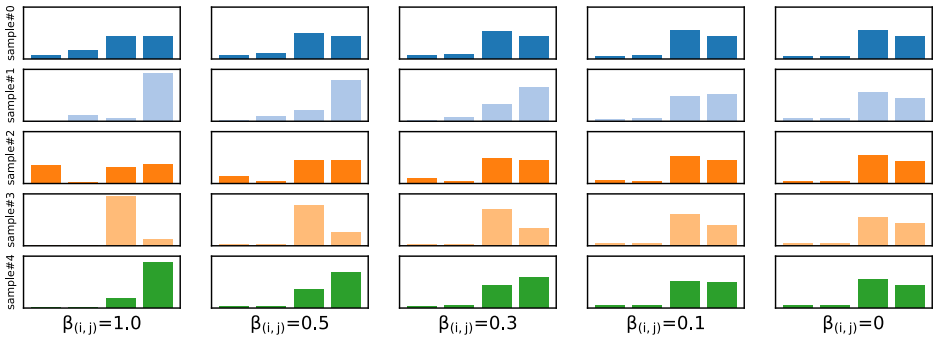

Figure 2: We sample $\boldsymbol{\alpha}^{(i,j)}$ five times from the concrete distribution (the first column) and the adaptive concrete distribution (the others) at $\tau = 1$. The class probabilities are $(0.05, 0.05, 0.5, 0.4)$, which are representative of the class probabilities of the learned distributions. Samples in every row share the same $\boldsymbol{\epsilon}^{(i,j)}$.

3.1 Adaptive Concrete Distribution

As shown in Eq. (7), we train the network weights based on the random structures sampled from the structure distribution. This makes $\boldsymbol{w}$ resist over-fitting, but also causes unignorable training challenges. Specifically, when the sampled structures have relatively high diversity, the network weights cannot fit all the structures well due to their limited capacity. Thus, in the test phase, the model will perform unsatisfactorily when sampling the network structures underfitted by the weights. We note that an analogous phenomenon was also observed by Mackay et al. (2019) in the gradient-based hyper-parameter optimization scenario: it is harmful that the distribution of the hyper-parameter has large variance because of the restricted flexibility of the approximation.

Therefore, to facilitate the convergence of the network weights, we expect to reduce the variance of the structure distribution. Specifically, we analyze the reparameterization procedure of the concrete distribution, and propose to multiply a tunable scalar $\boldsymbol{\beta}^{(i,j)}$ with $\boldsymbol{\epsilon}^{(i,j)}$ in the sampling:

$$\boldsymbol{\alpha}^{(i,j)} = g(\boldsymbol{\theta}^{(i,j)}, \boldsymbol{\beta}^{(i,j)}, \boldsymbol{\epsilon}^{(i,j)}) = \mathrm{softmax}((\boldsymbol{\theta}^{(i,j)} + \boldsymbol{\beta}^{(i,j)}\boldsymbol{\epsilon}^{(i,j)})/\tau). \tag{9}$$

Accordingly, we derive the log probability density of the adaptive concrete distribution which is slightly different from that of the concrete distribution (*see the detailed derivation in Appendix A*):

$$\begin{aligned}\log p(\boldsymbol{\alpha}^{(i,j)}|\boldsymbol{\theta}^{(i,j)}, \boldsymbol{\beta}^{(i,j)}) &= \log((K-1)!) + (K-1)\log\tau - (K-1)\log\boldsymbol{\beta}^{(i,j)} \\ &- \sum_{k=1}^{K}\log\boldsymbol{\alpha}_k^{(i,j)} + \sum_{k=1}^{K}\left[\frac{\boldsymbol{\theta}_k^{(i,j)} - \tau\log\boldsymbol{\alpha}_k^{(i,j)}}{\boldsymbol{\beta}^{(i,j)}}\right] - K * \mathrm{L\Sigma E}_{k=1}^{K}\left[\frac{\boldsymbol{\theta}_k^{(i,j)} - \tau\log\boldsymbol{\alpha}_k^{(i,j)}}{\boldsymbol{\beta}^{(i,j)}}\right],\end{aligned} \tag{10}$$

where $\mathrm{L\Sigma E}$ represents the log-sum-exp operation. With this, the last two terms of Eq. (6) can be estimated exactly.

Obviously, the adaptive concrete distribution degrades to the concrete distribution when $\boldsymbol{\beta}^{(i,j)} = 1$. As shown in Figure 2, sliding $\boldsymbol{\beta}^{(i,j)}$ from 1 to 0 decreases the diversity of the sampled structures gradually. Therefore, we should also keep $\boldsymbol{\beta}^{(i,j)}$ from being too small to avoid the over-fitting issue which the point-estimate structure (i.e., $\boldsymbol{\beta}^{(i,j)} = 0$) may suffer from. In practice, we choose to gradually reduce the sample variance along with the convergence of the weights, by decaying $\boldsymbol{\beta}^{(i,j)}$ from 1 to 0.5 with a linear schedule in the training.

3.2 Improvements of the Structure Learning Space

To further promote the convergence, we determine to improve the structure learning space used by NAS. The obtained structure learning space stabilizes the training and is substantially efficient.

Overall modification. To facilitate more effective information flow in the cell, we let the input of a cell (i.e., the output of the previous cell) be fixedly connected to all the internal nodes by $1\times1/3\times3$ convolutions in the classification/segmentation tasks. We only learn the connections between the

B internal nodes, as shown in Appendix D. The resulted nodes are concatenated with the input to form the output of the cell. Following DenseNet (Huang et al., 2017) and FC-DenseNet (Jégou et al., 2017), we limit the downsampling/upsampling modules to be the typical BN-ReLU-Conv-Pooling/ConvTranspose operations, to ease the learning of the network structure.

Batch normalization. NAS usually adopts the order of ReLU-Conv-BN in operations. However, in the searching stage, the learnable affine transformations in batch normalizations are always disabled to avoid the output rescaling issue (Liu et al., 2019). NAS does not suffer from this since it trains another network with learnable batch normalizations in the extra re-training stage. Instead, DBSN has to fix the issue to fit the data better. Thus, we propose to put a complete batch normalization in the front of the next layer. Namely, we adopt the BN-ReLU-Conv-BN convolutional layers, where the first BN has learnable affine parameters while the second one does not.

Candidate operations. In order to make the training more efficient, we remove the operations which are popular in NAS but unnecessary in DBSN, including all the 5×5 convolutions that can be replaced by stacked 3×3 convolutions, and all the pooling layers which are mainly used for the downsampling module. Then, the candidate operations in DBSN are: 3×3 separable convolutions, 3×3 dilated separable convolutions, identity and *zero*. We follow Liu et al. (2019) for the detailed settings of these operations.

Group operation. To obtain the j^{th} node in a cell, there are $(j-1)K$ operations from its predecessors to calculate, which can be organized into K groups according to the operation type. We note that the operations in a group are coincident and independent, so we advocate replacing them with a group operation (e.g., group convolution), which improves the efficiency significantly.

3.3 Discussion

Naturally, DBSN shows improved performance than the BNNs with weight uncertainty and even the advanced deep NNs, owing to its ability to adapt the network structure w.r.t. the data distribution. In the aspect of uncertainty, DBSN has more global randomness and is likely to ensemble more diverse predictions than the BNNs with weight uncertainty[3]. The diverse predictions complement each other in the ensembling, yielding more calibrated uncertainty, as testified in Section 5.3.

Compared to the differentiable NAS, DBSN is more charming since there is no need to split a considerable portion of the training set as validation set which reduces the data efficiency (Liu et al., 2019; Cai et al., 2019; Xie et al., 2019). In addition, DBSN jointly optimizes the structure and the weights, thus avoiding the harmful structure pruning operation and the laborious re-training stage.

4 Related Work

Learning flexible Bayesian models has long been the goal of the community (MacKay, 1992; Neal, 1995). The stochastic variational inference methods for Bayesian neural networks are particularly appealing owing to their analogy to the ordinary back-propagation (Graves, 2011; Blundell et al., 2015). More expressive distributions, such as matrix-variate Gaussians (Sun et al., 2017) or multiplicative normalizing flows (Louizos & Welling, 2017), have also been introduced to represent the posterior dependencies, but they are hard to train without heavy approximations. Recently, there is an increasing interest in developing Adam-like optimizers to perform natural-gradient variational inference for BNNs (Zhang et al., 2018; Bae et al., 2018; Khan et al., 2018). Despite enabling the scalability, these methods seem to demonstrate compromising performance compared to the state-of-the-art deep models. Interpreting the stochastic techniques of the deep models as Bayesian inference is also insightful (Gal & Ghahramani, 2016; Kingma et al., 2015; Teye et al., 2018; Mandt et al., 2017; Lakshminarayanan et al., 2017), but these methods still have relatively restricted and inflexible posterior approximations. Dikov & Bayer (2019) propose a unified Bayesian framework to infer the posterior of both the network weights and the structure, which is most similar to DBSN, but the network structure considered by them, namely layer size and network depth, only works for small networks but is essentially impractical for complicated deep models. Instead, we inherit the design

[3]Analogously, the neural networks with stochastic depth possibly make more diverse predictions than those with dropout (Huang et al., 2016).

of the structure learning space for NAS, and provide insightful techniques to boost the convergence, thus enabling effective Bayesian structure learning for deep neural networks.

Neural architecture search (NAS) has drawn tremendous attention in the past several years. Reinforcement learning (Zoph & Le, 2016; Zoph et al., 2018; Pham et al., 2018), evolution (Real et al., 2019) and Bayesian optimization (Kandasamy et al., 2018) have all been introduced to solve the challenge. More recently, differentiable NAS (Liu et al., 2019; Xie et al., 2019; Cai et al., 2019; Wu et al., 2019) shows promise since it reduces the prohibitive computational cost immensely. However, the existing differentiable NAS methods search the network structure in a meta-learning way (Finn et al., 2017), and need to re-train another network with the pruned compact structure after the searching. In contrast, DBSN unifies the learning of the weights and the structure in one training stage, alleviating the structure divergence and the inefficiency issues suffered by differentiable NAS.

5 EXPERIMENTS

To validate the structure learning ability and testify the predictive performance of DBSN, we first evaluate it on image classification and semantic segmentation tasks. For the estimation of the predictive uncertainty, we concern model calibration and generalization of the predictive uncertainty to adversarial examples as well as out-of-distribution samples, following existing works. We show that DBSN outperforms strong baselines in these tasks, shedding some light on practical Bayesian deep learning.

5.1 IMAGE CLASSIFICATION ON CIFAR-10 AND CIFAR-100

Setup. We set $B = 7, T = 4$ and $K = 4$, thus, $\boldsymbol{\alpha}$ consists of $7 \times 6/2 = 21$ sub-variables. The whole network is composed of 12 cells and 2 downsampling modules which have a channel compression factor of 0.4 and are located at the 1/3 and 2/3 depth. We employ a 3×3 convolution before the first cell and put a global average pooling followed by a fully connected (FC) layer after the last cell. The redundant operations all have 16 output channels. We set $\tau = \max(3 * \exp(-0.000015\mathrm{t}), 1)$ where t is the global training step. We initialize $\boldsymbol{w}$ and $\boldsymbol{\theta}$ following He et al. (2015) and Liu et al. (2019), respectively. The prior distributions of $\boldsymbol{\alpha}^{(i,j)}$ are set to be concrete distributions with uniform class probabilities. A momentum SGD with initial learning rate 0.1 (divided by 10 at 50% and 75% of the training procedure following (Huang et al., 2017)), momentum 0.9 and weight decay 10^{-4} is used to train the weights $\boldsymbol{w}$. We clip the gradient norm $||\nabla_{\boldsymbol{w}}||$ at 5. An Adam optimizer with learning rate 3×10^{-4}, momentum $(0.5, 0.999)$ is used to learn $\boldsymbol{\theta}$. We deploy the standard data augmentation scheme (mirroring/shifting) and normalize the data with the channel statistics. The whole training set is used for optimization. We train DBSN for 100 epochs with batch size 64, which takes one day on 4 GTX 1080-Tis. The implementation depends on PyTorch (Paszke et al., 2017) and the codes and are available online at `https://github.com/anonymousest/DBSN`.

Baselines. Besides comparison to the advanced deep models, we also design a series of baselines for fair comparisons. 1) **DBSN***: we substitute the concrete distribution for the adaptive concrete distribution. 2) **DBSN-1**: we use $T = 1$ sample in the gradient estimation. 3) **Fixed $\boldsymbol{\alpha}$**: we fix the structure of the network by setting the weight of every operation to be $1/K$. 4) **Dropout**: based on Fixed $\boldsymbol{\alpha}$, we further add dropout on every computational node with a drop rate of 0.2. 5) **Drop-path**: based on Fixed $\boldsymbol{\alpha}$, we further apply drop-path (Larsson et al., 2016) regularisation on the convolutional redundant operations with a path drop rate of 0.3. 6) **Random $\boldsymbol{\alpha}$**: we fix the distributions of $\boldsymbol{\alpha}^{(i,j)}$ as concrete distributions with uniform class probabilities and only train $\boldsymbol{w}$ with randomly sampled $\boldsymbol{\alpha}$. 7) **PE**: we view the structure as point estimates and train it as well as $\boldsymbol{w}$ simultaneously. 8) **DARTS**: we view the structure as point estimates but we train it on half of the training set while train $\boldsymbol{w}$ on the other half, resembling the first order DARTS (Liu et al., 2019). 9) **NEK-FAC**: we train a VGG16 network with weight uncertainty using the noisy EK-FAC (Bae et al., 2018) and the corresponding default settings. For PE and DARTS, the used structure optimizers have a weight decay of 10^{-3} to alleviate over-fitting.

When testing DBSN, DBSN*, DBSN-1, Random $\boldsymbol{\alpha}$, NEK-FAC, Dropout and Drop-path, we ensemble the predictive probabilities from 100 runs with randomly sampled structures/weights/drop masks/drop path masks to get the final predictions, respectively. We repeat every experiment 3 times and report the averaged error rate and standard deviation. The results are listed in Table 1.

Table 1: Comparison with competing baselines in terms of the number of parameters and test error rate. DBSN and its variants have 1.1 M parameters on CIFAR-100 due to a larger FC layer.

Method	Params (M)	CIFAR-10 (%)	CIFAR-100 (%)
ResNet (He et al., 2016a)	1.7	6.61	-
Stochastic Depth (Huang et al., 2016)	1.7	5.23	24.58
ResNet (pre-activation) (He et al., 2016b)	1.7	5.46	24.33
DenseNet (Huang et al., 2017)	1.0	5.24	24.42
DenseNet-BC (Huang et al., 2017)	0.8	**4.51**	**22.27**
NEK-FAC (Bae et al., 2018)	3.7	7.43	37.47
DBSN	1.0	$\mathbf{4.98} \pm 0.24$	$\mathbf{22.50} \pm 0.26$
DBSN*	1.0	5.22 ± 0.34	22.78 ± 0.19
DBSN-1	1.0	5.60 ± 0.17	23.44 ± 0.28
Fixed $\boldsymbol{\alpha}$	1.0	5.66 ± 0.24	24.27 ± 0.15
Random $\boldsymbol{\alpha}$	1.0	6.12 ± 0.12	23.60 ± 0.19
Dropout	1.0	5.83 ± 0.19	23.67 ± 0.28
Drop-path	1.0	5.77 ± 0.05	23.12 ± 0.13
PE	1.0	5.79 ± 0.34	24.19 ± 0.17
DARTS	1.0	8.91 ± 0.16	31.87 ± 0.12

Notably, DBSN demonstrates comparable performance with state-of-the-art deep neural networks. DBSN outperforms the powerful ResNet (He et al., 2016a) and DenseNet (Huang et al., 2017) with statistical evidence, and only presents modestly higher error rates than those of DenseNet-BC (Huang et al., 2017), which probably results from the usage of the expressive and efficient bottleneck layer in DenseNet-BC. This comparison highlights the practical value of DBSN.

Comparisons between DBSN and the baselines designed by ourselves are more insightful and convincing. 1) DBSN surpasses DBSN*, revealing the effectiveness of the adaptive concrete distribution. 2) DBSN-1 is remarkably worse than DBSN owing to the higher variance of the estimated gradients with only one sample. 3) Comparison of DBSN and Fixed $\boldsymbol{\alpha}$ validates that adapting the network structure w.r.t. the data distribution benefits the fitting of the model, resulting in substantially enhanced performance. 4) Random $\boldsymbol{\alpha}$, Dropout, and Drop-path train the networks with manually-designed untunable randomness, and hence are inferior to DBSN. 5) NEK-FAC gains rather compromising performance, with the powerful VGG16 architecture and one of the most advanced variational BNNs algorithms, suggesting us to prefer DBSN instead of the classic BNNs in the scenarios where the performance is a major concern. 6) PE and DARTS are two methods to learn the point-estimate network structure, both of which fall behind in terms of the test error. In particular, DARTS is much worse as it only trains the weights on half of the training set. This shows that DBSN is an appealing choice for effective neural structure learning with only one-stage training.

5.2 Semantic Segmentation on CamVid

To further verify that learning the network structure w.r.t. the data helps DBSN to obtain better performance than the standard NNs and BNNs, we apply DBSN to the challenging segmentation benchmark CamVid (Brostow et al., 2008). Our implementation is based on the brief FC-DenseNet framework (Jégou et al., 2017). Specifically, we only replace the original dense blocks with the structure-learnable cells, without introducing further advanced techniques from the semantic segmentation community, to figure out the performance gain only resulted from the learnable network structure. For the setup, we set $B = 5$ (same as the number of layers in every dense block of FC-DenseNet67) and $T = 1$, and learn two cell structures for the downsampling path and upsampling path, respectively. We use a momentum SGD with initial learning rate 0.01 (which decays linearly after 350 epochs), momentum 0.9 and weight decay 10^{-4} instead of the original RMSprop for better results. The other settings follow Jégou et al. (2017) and the classification experiments above. We also implement FC-DenseNet67 as a baseline. We present the results in Table 2 and Figure 3.

It is evident that DBSN surpasses the competing FC-DenseNet67 by a large margin while using fewer parameters. DBSN also demonstrates significantly better performance than the classic Bayesian SegNet which adopts MC dropout for uncertainty estimation. We emphasize this exper-

Table 2: Comparison of semantic segmentation performance on CamVid dataset. * indicates results from our implementation.

Method	Pretrained	Params (M)	Mean IoU	Global accuracy
SegNet (Badrinarayanan et al., 2015)	✓	29.5	46.4	62.5
Bayesian SegNet (Kendall et al., 2015)	✓	29.5	63.1	86.9
FC-DenseNet67 (Jégou et al., 2017)	✗	3.5	63.1*	90.4*
DBSN	✗	3.3	**65.4**	**91.4**

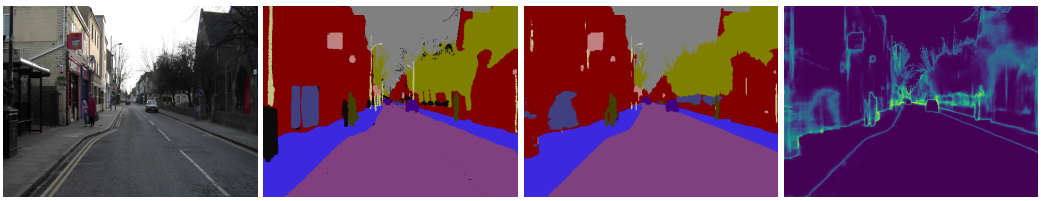

Figure 3: Visualization of the segmentation and uncertainty results of DBSN on CamVid. From left to right: original image, ground-truth segmentation, the estimated segmentation, and pixel-wise predictive uncertainty. The black color in ground-truth labels represents the background (void) class.

iment shows that the proposed framework is generally applicable. It is also worth noting that the uncertainty produced by DBSN is interpretable (see Figure 3): the edges of the objects and the regions which contain overlapping have substantially higher uncertainty than the other parts.

5.3 Estimation of Predictive Uncertainty

To validate that DBSN can provide promising predictive uncertainty, we evaluate it via calibration. We further examine the predictive uncertainty on adversarial examples and out-of-distribution (OOD) samples to test if the model *knows what it knows*. We also pay particular attention to the comparison between Drop-path and Dropout to double-check if more global randomness benefits predictive uncertainty more.

Calibration is orthogonal to the accuracy (Lakshminarayanan et al., 2017) and can be well estimated by the Expected Calibration Error (ECE) (Guo et al., 2017). Thus, we evaluate the trained models on the test set of CIFAR-10 and CIFAR-100 and calculate their ECE, as shown in Table 3. We also plot the reliability diagrams (Niculescu-Mizil & Caruana, 2005; Guo et al., 2017) of 4 typical methods DBSN, NEK-FAC, Dropout and Fixed $\boldsymbol{\alpha}$ in Appendix B, to provide a direct explanation of calibration. Unsurprisingly, DBSN achieves state-of-the-art calibration. DBSN outperforms the strong baselines, Dropout and NEK-FAC, and the superiority is especially obvious in the comparison to NEK-FAC. We also notice that Drop-path is better than Dropout in terms of ECE, supporting our hypothesis that the global randomness is more beneficial to the uncertainty than the local randomness.

To test the predictive uncertainty on the adversarial examples, we apply the fast gradient sign method (FGSM) (Goodfellow et al., 2014) to attack the trained models on CIFAR-10 and CIFAR-100 using the corresponding test samples[4]. Then we calculate the predictive entropy of the generated adversarial examples and depict the average entropy in Figure 4. As expected, the entropy of DBSN grows rapidly as the perturbation size increases, implying DBSN becomes pretty uncertain when encountering adversarial perturbations. By contrast, the change in entropy of Dropout and NEK-FAC is relatively moderate, which means that these methods are not as sensitive as DBSN to the adversarial examples. Besides, Drop-path is still better than Dropout, consistent with the conclusion above. We also note that Random $\boldsymbol{\alpha}$ has the highest predictive entropy. We speculate that this is because Random $\boldsymbol{\alpha}$ adopts the most diverse network structures (which results from the uniform class probabilities), and the ensemble of predictions from the corresponding networks is easier to be uniform.

[4]For DBSN, DBSN*, Random $\boldsymbol{\alpha}$, NEK-FAC, Dropout, and Drop-path, we attack using the ensemble of predictions from 30 stochastic runs and then we test the manipulated adversarial examples with 30 runs as well.

Table 3: Comparison of model calibration in terms of the Expected Calibration Error (ECE). Smaller is better.

Dataset	DBSN	DBSN*	Dropout	Drop-path	NEK-FAC	Fixed α	Random α	PE	DARTS
CIFAR-10	**0.0109**	0.0111	0.0150	0.0133	0.0434	0.0327	0.0458	0.0339	0.0488
CIFAR-100	**0.0599**	0.0677	0.0617	**0.0524**	0.1665	0.1259	0.0900	0.1240	0.1512

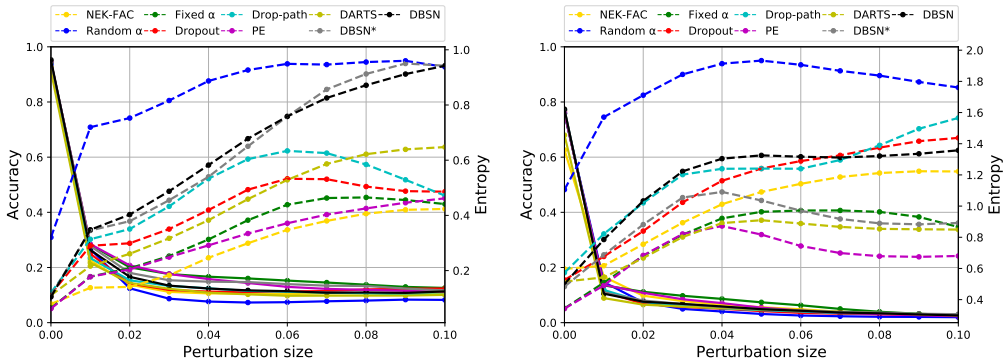

Figure 4: Accuracy (solid) and entropy (dashed) vary w.r.t. the adversarial perturbation size on CIFAR-10 (left) and CIFAR-100 (right).

We further attack with more powerful algorithms, e.g., the Basic Iterative Method (BIM) (Kurakin et al., 2016), and provide the results in Appendix C.

Moreover, we look into the entropy of the predictive distributions on OOD samples, to adequately evaluate the quality of uncertainty estimation. We use the trained models on CIFAR-10 and CIFAR-100, and take the samples from the test set of SVHN as OOD samples. We calculate their predictive entropy and draw the empirical CDF of the entropy in Figure 5, following Louizos & Welling (2017). The curve close to the bottom right corner is expected as it means most OOD samples have relatively large entropy (i.e., low prediction confidence). Obviously, DBSN demonstrates comparable or even better results than the competing methods like Dropout and NEK-FAC. In addition, Drop-path attains substantially improved results than Dropout. Analogous to the experiments on adversarial examples, Random $\boldsymbol{\alpha}$ provides impressive predictive uncertainty on the OOD samples.

In conclusion, DBSN consistently delivers state-of-the-art predictive uncertainty in various scenarios, which validates the effectiveness of uncertainty on network structure. We also confirm that more global/structured randomness induced in networks helps to improve the uncertainty estimation significantly.

5.4 Rethinking of the One-shot NAS

One-shot NAS (Bender et al., 2018; Guo et al., 2019) first trains the weights of a super network and then searches for a good structure given the weights. This avoids the bias induced by the gradient-based joint optimization of the differentiable NAS. However, we argue that the super network trained with the fixed (Bender et al., 2018) or uniformly sampled (Guo et al., 2019) network structures cannot flexibly focus its capacity on the most crucial operations, harming the subsequent searching. To this end, we have conducted a set of experiments to check whether dynamically adjusting the network structure at the stage of weight training helps to find better network structures eventually. Observing that DBSN trains a super network with adaptive network structures and Random $\boldsymbol{\alpha}$ trains a super network with unadjustable structures (similar to the uniform sampling used by Guo et al. (2019)), we choose to search for the optimal structure distributions based on the trained weights from DBSN and Random $\boldsymbol{\alpha}$[5]. After searching, we train new networks with the searched structure distributions (fixed in the training) from scratch, and then test their performance. The results are

[5]We initialize $\boldsymbol{\theta}^{(i,j)}$ randomly and initialize $\boldsymbol{\beta}^{(i,j)}$ with 1. Given the fixed network weights, we optimize $\boldsymbol{\theta}^{(i,j)}$ and $\boldsymbol{\beta}^{(i,j)}$ by gradient descent. The searching lasts for 20 epochs.

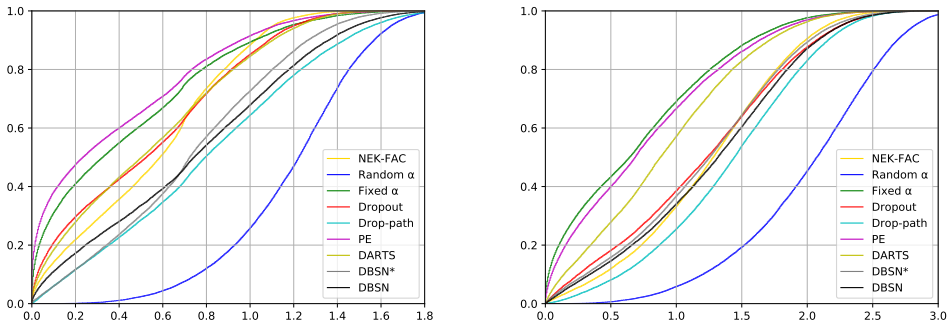

Figure 5: Empirical CDF for the entropy of the predictive distributions on SVHN dataset of models trained on CIFAR-10 (left) and CIFAR-100 (right). The curves that are closer to the bottom right corner are better.

Table 4: Comparison of the searched structure distributions based on the trained network weights from DBSN and Random $\boldsymbol{\alpha}$.

	DBSN	**Random $\boldsymbol{\alpha}$**
Test error (%)	**5.46**	5.98

shown in Table 4. It is clear that the searched structure distribution based on the weights trained by DBSN outperforms the other one significantly, supporting our hypotheses. Therefore, we propose to reasonably adapt the structure in the weight-training stage of one-shot NAS, which drives the most useful operations to be optimized thoroughly and eventually yields more powerful network structures.

5.5 Visualization of the Learned Structure Distributions

We visualize the learned structure distributions in Appendix D. The structure distributions for different tasks look quite different, which implies that the structures are learned in a way that accounts for the specific characteristics in the data.

6 Conclusion

In this work, we have introduced a novel Bayesian structure learning approach for deep neural networks. The proposed DBSN draws the inspiration from the network design of NAS and models the network structure as Bayesian variables. Stochastic variational inference is employed to jointly learn the network weights and the distribution of the network structure. We further develop the adaptive concrete distribution and improve the structure learning space to facilitate the convergence of the whole model. Empirically, DBSN has revealed impressive performance on the discriminative learning tasks, surpassing the advanced deep models, and presented state-of-the-art predictive uncertainty in various scenarios. In conclusion, DBSN provides a more practical way for Bayesian deep learning, without compromise between the predictive performance and the Bayesian uncertainty.

There are two major directions for future work. On one hand, the current DBSN is not efficient enough, owing to the redundant operations involved in the computations. We also need to estimate the gradients using $T > 1$ sampled structures for better convergence, which increases the computational overhead further. Therefore, some strategies need to be discovered to make DBSN more efficient. On the other hand, DBSN still has a relatively restricted structure learning space. Thus, more operations can be introduced and more global network structures can be learned in future work.

REFERENCES

Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

Juhan Bae, Guodong Zhang, and Roger Grosse. Eigenvalue corrected noisy natural gradient. *arXiv preprint arXiv:1811.12565*, 2018.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pp. 549–558, 2018.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pp. 1613–1622, 2015.

Gabriel J Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *European conference on computer vision*, pp. 44–57. Springer, 2008.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HylVB3AqYm`.

Georgi Dikov and Justin Bayer. Bayesian learning of neural network architectures. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 730–738, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

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

Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Structured variational learning of bayesian neural networks with horseshoe priors. In *International Conference on Machine Learning*, pp. 1739–1748, 2018.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Alex Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pp. 2348–2356, 2011.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1321–1330. JMLR. org, 2017.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

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*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Geoffrey Hinton and Drew Van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. Citeseer, 1993.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 11–19, 2017.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pp. 2016–2025, 2018.

Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In *International Conference on Machine Learning*, pp. 2616–2625, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pp. 6402–6413, 2017.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=S1eYHoC5FX`.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances in neural information processing systems*, pp. 2378–2386, 2016.

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2218–2227. JMLR. org, 2017.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Matthew Mackay, Paul Vicol, Jonathan Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1eEG20qKQ`.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907, 2017.

Radford M Neal. *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD thesis, Citeseer, 1995.

Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pp. 625–632. ACM, 2005.

Kazuki Osawa, Siddharth Swaroop, Anirudh Jain, Runa Eschenhagen, Richard E Turner, Rio Yokota, and Mohammad Emtiyaz Khan. Practical deep learning with bayesian principles. *arXiv preprint arXiv:1906.02506*, 2019.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Tim Pearce, Mohamed Zaki, Alexandra Brintrup, and Andy Neely. Expressive priors in bayesian neural networks: Kernel combinations and periodic functions. *arXiv preprint arXiv:1905.06076*, 2019.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pp. 4092–4101, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019.

Jiaxin Shi, Shengyang Sun, and Jun Zhu. Kernel implicit variational inference. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1l4eQW0Z.

Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning structured weight uncertainty in bayesian neural networks. In *Artificial Intelligence and Statistics*, pp. 1283–1292, 2017.

Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. Functional variational bayesian neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rkxacs0qY7.

Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. In *International Conference on Machine Learning*, pp. 4914–4923, 2018.

Ziyu Wang, Tongzheng Ren, Jun Zhu, and Bo Zhang. Function space particle optimization for bayesian neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BkgtDsCcKQ.

Yu Weng, Tianbao Zhou, Yujie Li, and Xiaoyu Qiu. Nas-unet: Neural architecture search for medical image segmentation. *IEEE Access*, 7:44247–44257, 2019.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rylqooRqK7.

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, pp. 5847–5856, 2018.

Jingwei Zhuo, Chang Liu, Jiaxin Shi, Jun Zhu, Ning Chen, and Bo Zhang. Message passing stein variational gradient descent. In *International Conference on Machine Learning*, pp. 6013–6022, 2018.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A Derivation of the Log Probability Density of the Adaptive Concrete Distribution

For clear expression, We simply denote $\boldsymbol{\alpha}^{(i,j)}$, $\boldsymbol{\theta}^{(i,j)}$, $\boldsymbol{\beta}^{(i,j)}$ and $\boldsymbol{\epsilon}^{(i,j)}$ as $\boldsymbol{\alpha}$, $\boldsymbol{\theta}$, β and $\boldsymbol{\epsilon}$, respectively. Let $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{\theta})$. Consider

$$\boldsymbol{\alpha}_k = \frac{\exp((\boldsymbol{\theta}_k + \beta\boldsymbol{\epsilon}_k)/\tau)}{\sum_{i=1}^K \exp((\boldsymbol{\theta}^i + \beta\boldsymbol{\epsilon}^i)/\tau)} = \frac{\exp((\log \boldsymbol{p}_k + \beta\boldsymbol{\epsilon}_k)/\tau)}{\sum_{i=1}^K \exp((\log \boldsymbol{p}^i + \beta\boldsymbol{\epsilon}^i)/\tau)}.$$

Let $\boldsymbol{z}_k = \log \boldsymbol{p}_k + \beta\boldsymbol{\epsilon}_k = \log \boldsymbol{p}_k - \beta \log(-\log(\boldsymbol{u}_k))$, where $\boldsymbol{u}_k \sim \mathcal{U}(0,1)$ i.i.d.. It has density

$$\frac{1}{\beta}\boldsymbol{p}_k^{1/\beta} \exp(-\frac{\boldsymbol{z}_k}{\beta}) \exp(-\boldsymbol{p}_k^{1/\beta} \exp(-\frac{\boldsymbol{z}_k}{\beta})).$$

We denote $c = \sum_{i=1}^K \exp(\boldsymbol{z}_i/\tau)$, then $\boldsymbol{\alpha}_k = \exp(\boldsymbol{z}_k/\tau)/c$. We consider this transformation:

$$F(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K) = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{K-1}, c),$$

which has the following inverse transformation:

$$F^{-1}(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{K-1}, c) = (\tau(\log \boldsymbol{\alpha}_1 + \log c), \ldots, \tau(\log \boldsymbol{\alpha}_K + \log c)),$$

whose Jacobian has the determinant (refer to the derivation of the concrete distribution (Maddison et al., 2016)):

$$\frac{\tau^K}{c\prod_{i=1}^K \boldsymbol{\alpha}_i}.$$

Multiply this with the density of $\boldsymbol{z}$, we get the density

$$\frac{\tau^K \prod_{i=1}^K \frac{1}{\beta}\boldsymbol{p}_i^{1/\beta} \exp(-\frac{\tau(\log \boldsymbol{\alpha}_i + \log c)}{\beta}) \exp(-\boldsymbol{p}_i^{1/\beta} \exp(-\frac{\tau(\log \boldsymbol{\alpha}_i + \log c)}{\beta}))}{c\prod_{i=1}^K \boldsymbol{\alpha}_i}.$$

Let $r = \log c$, then apply the change of variables formula, we obtain the density:

$$\frac{\tau^K \prod_{i=1}^K \boldsymbol{p}_i^{1/\beta}}{\beta^K \prod_{i=1}^K \boldsymbol{\alpha}_i^{(1+\tau/\beta)}} \exp(-\frac{K\tau r}{\beta}) \exp(-\sum_{i=1}^K (\boldsymbol{p}_i \boldsymbol{\alpha}_i^{-\tau})^{1/\beta} \exp(-\frac{\tau r}{\beta})).$$

We use γ to substitute $\log \sum_{i=1}^K (\boldsymbol{p}_i \boldsymbol{\alpha}_i^{-\tau})^{1/\beta}$, then get:

$$\frac{\tau^K \prod_{i=1}^K \boldsymbol{p}_i^{1/\beta}}{\exp(\gamma)\beta^K \prod_{i=1}^K \boldsymbol{\alpha}_i^{(1+\tau/\beta)}} \exp(\gamma - \frac{K\tau r}{\beta}) \exp(-\exp(\gamma - \frac{\tau r}{\beta})).$$

Naturally, we can integrate out r, and get:

$$\begin{aligned}
&\frac{\tau^K \prod_{i=1}^K \boldsymbol{p}_i^{1/\beta}}{\exp(\gamma)\beta^K \prod_{i=1}^K \boldsymbol{\alpha}_i^{(1+\tau/\beta)}} \left[\frac{\beta}{\tau} \exp(\gamma - K\gamma)\Gamma(K)\right] \\
=&\frac{\tau^{K-1} \prod_{i=1}^K \boldsymbol{p}_i^{1/\beta}}{\beta^{K-1} \prod_{i=1}^K \boldsymbol{\alpha}_i^{(1+\tau/\beta)}} \exp(-K\gamma)\Gamma(K) \\
=&\frac{((K-1)!)\tau^{K-1}}{\beta^{K-1} \prod_{i=1}^K \boldsymbol{\alpha}_i} \times \frac{\prod_{i=1}^K (\boldsymbol{p}_i \boldsymbol{\alpha}_i^{-\tau})^{1/\beta}}{(\sum_{i=1}^K (\boldsymbol{p}_i \boldsymbol{\alpha}_i^{-\tau})^{1/\beta})^K}.
\end{aligned}$$

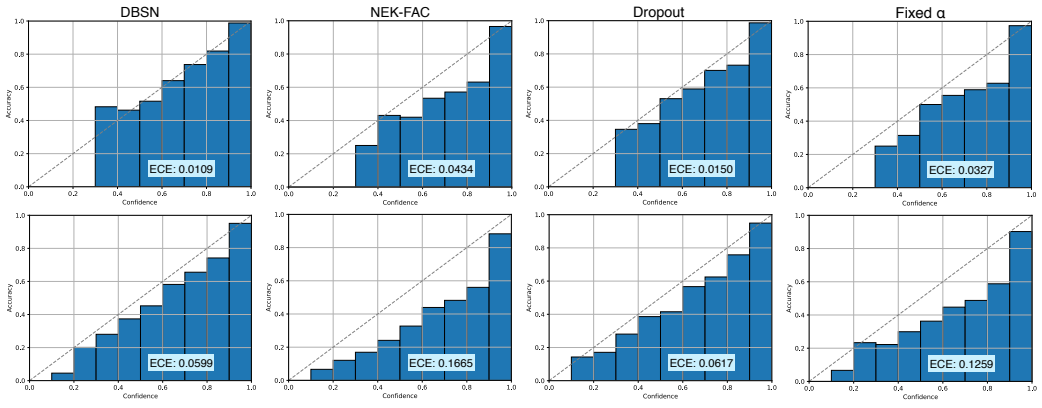

Figure 6: Reliability diagrams for DBSN, NEK-FAC, Dropout and Fixed $\boldsymbol{\alpha}$ (from left to right) on CIFAR-10 (top row) and CIFAR-100 (bottom row). The bars aligning alone the diagonal are expected.

Then, the log density is:

$$\log((K-1)!) + (K-1)\log\frac{\tau}{\beta} - \sum_{i=1}^{K}\log\boldsymbol{\alpha}_i + \sum_{i=1}^{K}\frac{\log\boldsymbol{p}_i - \tau\log\boldsymbol{\alpha}_i}{\beta} - K * \mathrm{L}\sum_{i=1}^{K}\mathrm{E}\frac{\log\boldsymbol{p}_i - \tau\log\boldsymbol{\alpha}_i}{\beta}$$

$$= \log((K-1)!) + (K-1)\log\frac{\tau}{\beta} - \sum_{i=1}^{K}\log\boldsymbol{\alpha}_i + \sum_{i=1}^{K}\frac{\boldsymbol{\theta}_i - \tau\log\boldsymbol{\alpha}_i}{\beta} - K * \mathrm{L}\sum_{i=1}^{K}\mathrm{E}\frac{\boldsymbol{\theta}_i - \tau\log\boldsymbol{\alpha}_i}{\beta},$$

which is equal to Eq. (10).

B More Results for Calibration

We plot the reliability diagrams of 4 typical methods, which represent the deep BNN with structure uncertainty, the classic BNN with weight uncertainty, the deterministic NN with MC dropout and the standard NN, respectively, in Figure 6. Obviously, DBSN has better reliability diagrams than NEK-FAC and Dropout, proving the effectiveness of the uncertainty on network structure.

C Attack with BIM

We perform an adversarial attack using BIM algorithm. Concretely, we set the number of iteration to be 3 and set the perturbation size in every step to be 1/3 of the whole perturbation size. The experiments mainly focus on the models trained on CIFAR-10. Figure 7 shows the results. Random $\boldsymbol{\alpha}$, DBSN* and DBSN have increasing entropy when the perturbation size changes from 0 to 0.01, but all the other approaches are attacked successfully with entropy dropping. However, strictly, only the Random $\boldsymbol{\alpha}$ at perturbation size 0.01 provides useful predictive uncertainty, and we can use the entropy to reject the predictions. Therefore, we have to agree that BIM is powerful enough to break all the methods, including DBSN. So we advise adjusting DBSN accordingly (e.g., employing adversarial training, using more robust loss) if we want to use DBSN to defend the adversarial attacks.

D Visualization of the Learned Structures

We visualize the learned structure distributions on different tasks in Figure 8, Figure 9, Figure 10 and Figure 11 (we do not draw the *zero* operation). The structure distributions learned on different tasks look different, validating that DBSN can adapt the network structure according to the data distribution flexibly.

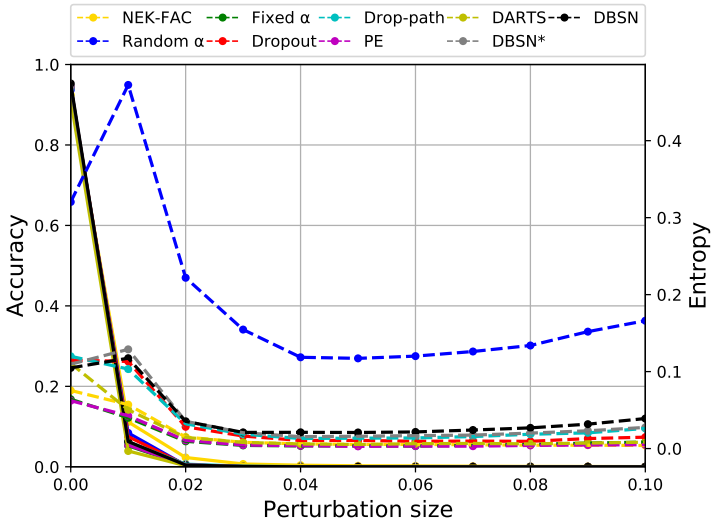

Figure 7: Accuracy (solid) vs entropy (dashed) as a function of the adversarial perturbation size on CIFAR-10. Attack with BIM.

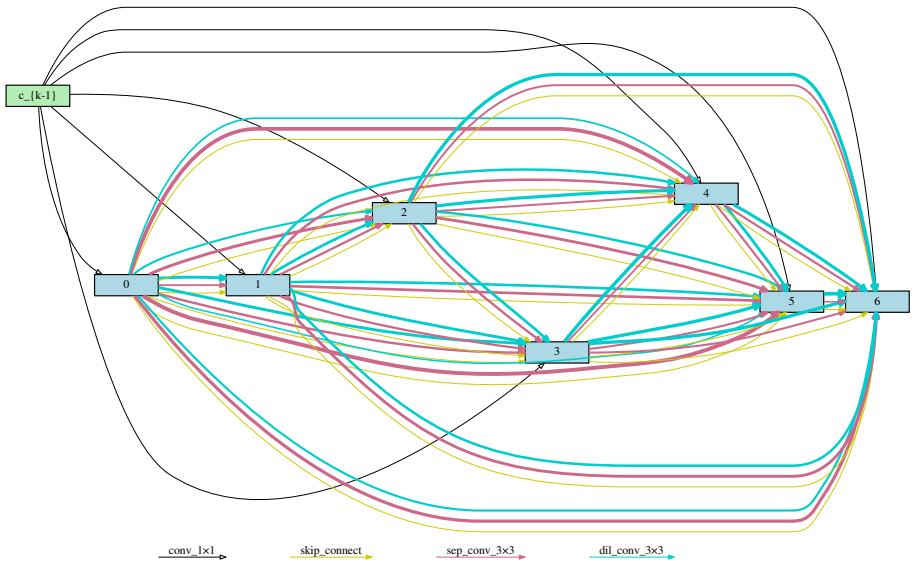

Figure 8: Structure of the cell learned on CIFAR-10. The pen width of an edge implies the sampling probability of its corresponding operation.

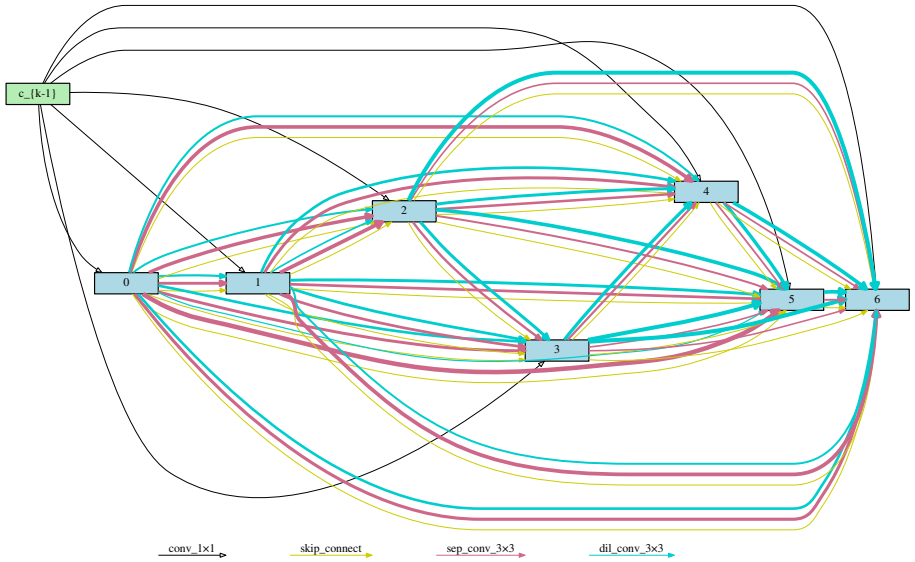

Figure 9: Structure of the cell learned on CIFAR-100. The pen width of an edge implies the sampling probability of its corresponding operation.

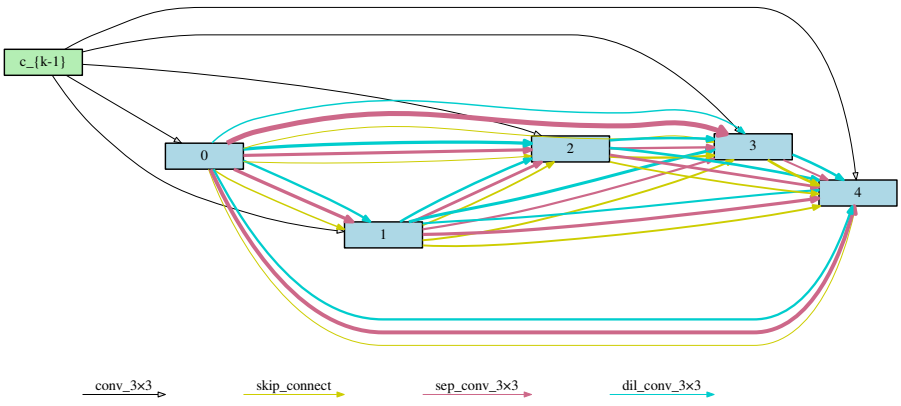

Figure 10: Structure of the cell learned on CamVid (in the downsampling path). The pen width of an edge implies the sampling probability of its corresponding operation.

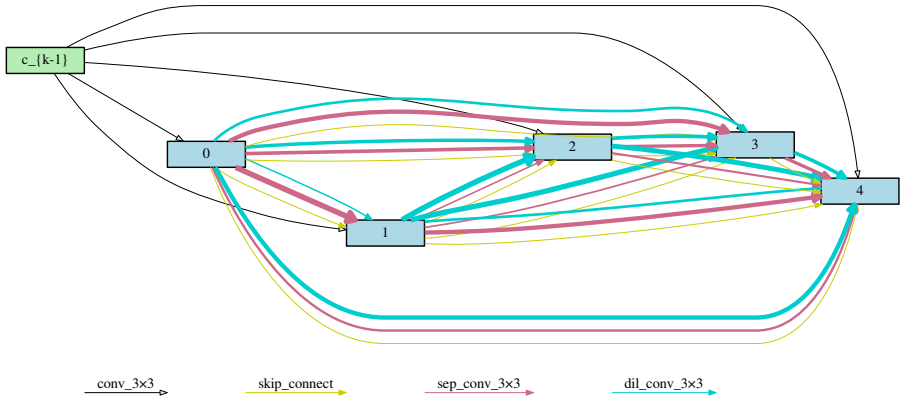

Figure 11: Structure of the cell learned on CamVid (in the upsampling path). The pen width of an edge implies the sampling probability of its corresponding operation.