Bayesian Nested Neural Networks for Uncertainty Calibration and Adaptive Compression

Yufei Cui Ziquan Liu Qiao Li Yu Mao Antoni B. Chan Chun Jason Xue

Abstract

Nested networks or slimmable networks are neural networks whose architectures can be adjusted instantly during testing time, e.g., based on computational constraints. Recent studies have focused on a "nested dropout" layer, which is able to order the nodes of a layer by importance during training, thus generating a nested set of sub-networks that are optimal for different configurations of resources. However, the dropout rate is fixed as a hyper-parameter over different layers during the whole training process. Therefore, when nodes are removed, the performance decays in a human-specified trajectory rather than in a trajectory learned from data. Another drawback is the generated sub-networks are deterministic networks without well-calibrated uncertainty. To address these two problems, we develop a Bayesian approach to nested neural networks. We propose a variational ordering unit that draws samples for nested dropout at a low cost, from a proposed Downhill distribution, which provides useful gradients to the parameters of nested dropout. Based on this approach, we design a Bayesian nested neural network that learns the order knowledge of the node distributions. In experiments, we show that the proposed approach outperforms the nested network in terms of accuracy, calibration, and out-of-domain detection in classification tasks. It also outperforms the related approach on uncertainty-critical tasks in computer vision.

1. Introduction

Modern deep neural networks (DNNs) have achieved great success in fields of computer vision and related areas. In the meantime, they are experiencing rapid growth in model size and computation cost, which makes it difficult to deploy on diverse hardware platforms. Recent works study how to develop a network with flexible size during test time [20, 50, 49, 4, 6, 47], to reduce the cost in designing [44], training [21], compressing [13] and deploying [36] a DNN on various platforms. As these networks are often composed of a nested set of smaller sub-networks, we refer to them as *nested nets* in this paper. As many problems are safety-critical, such as object recognition [9, 12], medical-image segmentation [24, 18] and crowd counting [33, 45], the adopted DNNs are required to provide well-calibrated uncertainty in addition to high prediction performance, as erroneous predictions could result in disastrous consequences. However, the measure of uncertainty was not considered in previous designs of nested nets, which leads to over- or under-confident predictions.

One basis for creating nested nets is to order the network components (e.g., convolution channels) such that less important components can be removed first when creating the sub-network. A unit for neural networks, *nested dropout*, was developed to order the latent feature representation for the encoder-decoder models [37, 1]. Specifically, a discrete distribution is assigned over the indices of the representations, and the operation of nested dropout samples an index then drops the representations with larger indices. Recent studies show that the *nested dropout* is able to order the network components during training such that *nested nets* can be obtained [6, 7]. The ordering layout is applicable to different granularity levels of network components: single weights, groups of weights, convolutional channels, residual blocks, network layers, and even quantization bits. We refer to the partitions of the network components as *nodes* in this paper. However, the probability that an index is sampled is specified by hand as a hyperparameter, and does not change during training. Thus, the importance of a node is pre-determined by hand rather than learned from data.

To enhance predictive uncertainty and to allow the dropout rate to be learned, in this paper, we propose a fully Bayesian treatment for constructing nested nets. We propose a new nested dropout, based on a chain of interdependent Bernoulli variables. The chain simulates the Bernoulli trials and can be understood as a special case of a two-state Markov chain, which intuitively generates order information. To save the time cost for sampling during training, we propose a variational ordering unit that approximates the chain, and an approximate posterior based on a novel *Downhill* distribution built on Gumbel Softmax [17, 29]. This allows efficient sampling of the multivariate *ordered*

Department of Computer Science, City University of Hong Kong. Correspondence to: Yufei Cui. Email: yufeicui92@gmail.com.

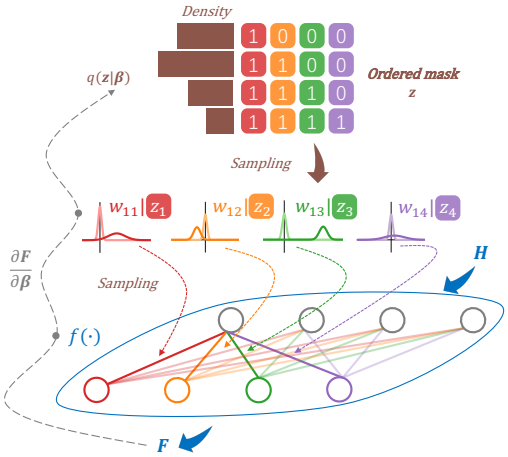

Figure 1: Sampling process in a layer for calculating the data log-likelihood (Eq. 8). A fully connected layer $f(\cdot)$ takes $\mathbf{H}$ as an input and outputs $\mathbf{F}$. The variational ordering unit $q(\mathbf{z}|\boldsymbol{\beta})$ generates ordered mask $\mathbf{z} = [z_j]_j$. Nodes w_{ij}'s with the same color share an element z_j. The gradient through stochastic nodes $\frac{\partial \mathbf{F}}{\partial \boldsymbol{\beta}}$ can be estimated efficiently, to update the importance $\boldsymbol{\beta}$.

mask, and provides useful gradients to update the importance of the nodes.

Based on the proposed ordering units, a Bayesian nested neural network is constructed, where the independent distributions of nodes are interconnected with the ordering units. A mixture model prior is placed over each node, while the model selection is determined by the ordering units (Fig. 1). A variational inference problem is formulated and resolved, and we propose several methods to simplify the sampling and calculation of the regularization term. Experiments show that our model outperforms the deterministic nested models in any sub-network, in terms of classification accuracy, calibration and out-of-domain detection. It also outperforms other uncertainty calibration methods on uncertainty-critical vision tasks, e.g., probabilistic U-Net on medical segmentation with noisy labels.

In summary, the contributions of this paper are:

- We propose a variational nested dropout unit with a novel pair of prior and posterior distributions.
- We propose a novel Bayesian nested neural network that can generate large sets of uncertainty-calibrated sub-networks. The formulation can be viewed as a generalization of *ordered* ℓ_0 regularization over the sub-networks.
- To our knowledge, this is the first work that considers uncertainty calibration and learned the importance of network components in nested neural networks.

2. Variational Nested Dropout

We first review nested dropout, and then propose our Bayesian ordering unit and variational approximation.

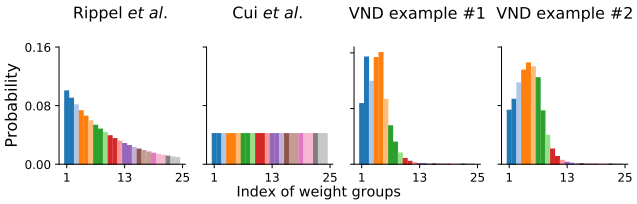

Figure 2: The probability of tail index being sampled in different nested dropout realizations. Rippel *et al.* [37] and Cui *et al.* [6] adopt Geometric and Categorical distributions, which are static over different layers and the learning process. The proposed variational nested dropout (VND) learns the importances of nodes from data. The two examples are from two different layers in a Bayesian nested neural network.

2.1. A review of nested dropout

The previous works [37] that order the representations use either Geometric or Categorical distributions to sample the last index of the kept units, then drop the neurons with indices greater than it. Specifically, the distribution $p_B(\cdot)$ is assigned over the representation indices $1, \dots, K$. The nested/ordered dropout operation proceeds as follows:

1. *Tail sampling:* A tail index is sampled $b \sim p_B(\cdot)$ that represents the last element be kept.
2. *Ordered dropping:* The elements with indices $b+1, \dots, K$ are dropped.

We also refer to this operation as an *ordering unit* as the representations are sorted in order.

In [37], which focuses on learning ordered representations, this operation is proved to exactly recover PCA with a one-layer neural network. Cui *et al.* [6] shows this operation, when applied to groups of neural network weights or quantization bits, generates nested sub-networks that are optimal for different computation resources. They further prove that increasing from a smaller sub-network to a larger one maximizes the incremental information gain. A large network only needs to be trained once with ordered dropout, yielding a set of networks with varying sizes for deployment. However, the above methods treat the ordered dropout rate as a hyper-parameter, and hand-tuning the dropout rate is tedious and may lead to suboptimal performance of the sub-networks, as compared to learning this hyperparameter from the data. As illustrated in Fig. 2, the previous works use hand-specified parameters for nested dropout, which freezes the importance of the network components over different layers during training.

A common practice for regular Bernoulli dropout is to treat the dropout rate as a variational parameter in Bayesian neural networks [8]. To find the optimal dropout rate, grid-search is first adopted [9], whose complexity grows exponentially with the number of dropout units. To alleviate the cost of searching, a continuous relaxation of the discrete dropout is proposed by which the dropout rate can be optimized directly [10], improving accuracy and uncertainty, while keeping a low training time. However, for *nested dropout*, two aspects are unclear: 1) how to take a full

Bayesian treatment with nested dropout units; 2) how the relaxation can be done for these units or how the gradients can be back-propagated to the parameters of $p_B(\cdot)$.

2.2. Bayesian Ordering Unit

The conventional nested dropout uses a Geometric distribution to sample the tail b, $p_B(b=i) = (1-\pi)^i\pi$, for $i \in \{1, \dots, K\}$. By definition, the Geometric distribution models the probability that the i-th trial is the first "success" in a sequence of independent Bernoulli trials. In the context of slimming neural networks, a "failure" of a Bernoulli trial indicates that node is kept, while a "success" indicates the tail index, where this node is kept and all subsequent nodes are dropped. Thus, π is the conditional probability of a node being a tail index, given the previous node is kept.

Sampling from the Geometric only generates the tail index of the nodes to be kept. A hard selection operation of ordered dropping is required to drop the following nodes. The ordered dropping can be implemented using a set of ordered mask vectors $\mathcal{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_K\}$, where $\mathbf{v}_j$ consists of j ones followed by $K-j$ zeros, $\mathbf{v}_j = [\underbrace{1, \dots, 1}_{j}, \underbrace{0, \dots, 0}_{K-j}]$. Given the sampled tail index $b \sim p_B(\cdot)$, the appropriate mask $\mathbf{v}_b$ is selected and applied to the nodes (e.g., multiplying the weights). However, as the masking is a non-differentiable transformation and does not provide a well-defined probability distribution, the nested dropout parameters cannot be learned using this formulation.

To find a more natural prior for the nodes, we propose to use a chain of Bernoulli variables to *directly* model the distribution of the *ordered masks*. Let the set of binary variables $\mathbf{z} = [z_1, \dots, z_K]$ represent the random ordered mask. Specifically, we model the conditional distributions with Bernoulli variables,

$$\begin{aligned} &p(z_1 = 1) = \pi_1, && p(z_1 = 0) = 1 - \pi_1, \\ &p(z_i = 1 | z_{i-1} = 1) = \pi_i, && p(z_i = 0 | z_{i-1} = 1) = 1 - \pi_i, \\ &p(z_i = 1 | z_{i-1} = 0) = 0, && p(z_i = 0 | z_{i-1} = 0) = 1, \end{aligned} \tag{1}$$

where π_i is the conditional probability of keeping the node given the previous node is kept, and $\pi_1 = 1$ (the first node is always kept). Note that we also allow different probabilities π_i for each z_i. The marginal distribution of z_i is

$$p(z_i = 1) = \prod_{k=1}^{i} \pi_k, \quad p(z_i = 0) = 1 - \prod_{k=1}^{i} \pi_k. \tag{2}$$

A property of this chain is that if 0 occurs at the i-th position, the remaining elements with indices $i+1, \dots, K$ become 0. That is, sampling from this chain generates an ordered mask, which can be directly multiplied on the nodes to realize *ordered dropping*. Another benefit is that applying a continuous relaxation [10] of the Bernoulli variables in the chain allows its parameters $\boldsymbol{\pi}$ to be optimized.

However, the sampling of $\mathbf{z}$ requires stepping through each element z_i, which has complexity $\mathcal{O}(K)$, and is thus not scalable in modern DNNs where K is large. Thus we apply the variational inference framework, while treating $p(\mathbf{z})$ as the prior of the ordered mask in our Bayesian treatment. One challenge is to find a tractable variational distribution $q(\mathbf{z})$ that approximates the true posterior and is easy to compute. Another challenge is to define a $q(\mathbf{z})$ that allows efficient re-parameterization, so that the gradient of the parameter of $q(\mathbf{z})$ can be estimated with low variance.

2.3. Variational Ordering Unit

We propose a novel *Downhill* distribution based on Gumbel Softmax distribution [17, 29] that generates the ordered mask $\mathbf{z}$.

Definition 1 *Downhill Random Variables (r.v.). Let the temperature parameter $\tau \in (0, \infty)$. An r.v. $\mathbf{z}$ has a Downhill distribution $\mathbf{z} \sim \text{Downhill}(\boldsymbol{\beta}, \tau)$, if its density is:*

$$\begin{aligned} &q(z_1, \dots, z_K) \\ =&\Gamma(K)\tau^{K-1}\left[\sum_{i=1}^{K} \frac{\beta_i}{(z_{i-1} - z_i)^\tau}\right]^{-K} \prod_{i=1}^{K} \frac{\beta_i}{(z_{i-1} - z_i)^{\tau+1}}, \end{aligned} \tag{3}$$

where $\boldsymbol{\beta} = [\beta_1, \dots, \beta_K]$ are the probabilities for each dimension.

Two important properties of Downhill distributions are:

- **Property 1.** If $\mathbf{c} \sim \text{Gumbel\_softmax}(\tau, \beta, \epsilon_z)$[1], then $z_i = 1 - \text{cumsum}'_i(\mathbf{c})$, where $\mathbf{e}$ is a K-dimensional vector of ones, and $\text{cumsum}'_i(\mathbf{c}) = \sum_{j=0}^{i-1} c_j$. $c_0 \coloneqq 1$. ϵ_z is a standard uniform variable.
- **Property 2.** When $\tau \to 0$, sampling from the Downhill distribution reduces to discrete sampling, where the sample space is the set of ordered mask vectors $\mathcal{V}$. The approximation of the Downhill distribution to the Bernoulli chain can be calculated in closed-form.

Property 1 shows the sampling process of the Downhill distribution. We visualize the Downhill samples in Fig. 3. As each multivariate sample has a shape of a long descent from left to right, we name it *Downhill* distribution. The temperature variable τ controls the sharpness of the downhill or the smoothness of the step at the tail index. When τ is large, the slope is gentle in which case no nodes are dropped, but the less important nodes are multiplied with a factor less than 1. When $\tau \to 0$, the shape of the sample becomes a cliff which is similar to the prior $p(\mathbf{z})$ on ordered masks, where the less important nodes are dropped (i.e., multiplied by 0). Property 1 further implies the gradient $\frac{\partial}{\partial \boldsymbol{\beta}} \mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\beta}}(\mathbf{z})}[\zeta(\mathbf{z})]$

[1]For Gumbel-softmax sampling, we first draw $g_1 \dots g_K$ from $\text{Gumbel}(0, 1)$, then calculate $c_i = \text{softmax}(\frac{\log(\beta_i) + g_i}{\tau})$. The samples of $\text{Gumbel}(0, 1)$ can be obtained by first drawing $\epsilon_z \sim \text{Uniform}(0, 1)$ then computing $g = -\log(-\log(\epsilon_z))$.

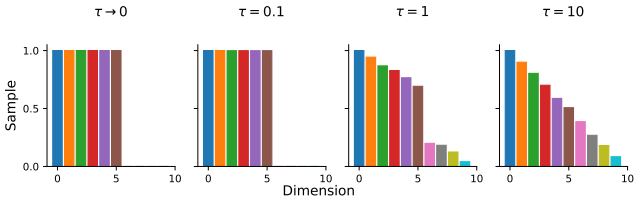

Figure 3: The multivariate Downhill samples under different temperatures τ. When $\tau \to 0$, a clear cliff is observed as the dimension increases, which is beneficial for differentiating important or unimportant nodes. As τ increases, the shape becomes a slope where the gaps between important/unimportant nodes are smoother, which is beneficial for training.

can be estimated with low variance, for a cost function $\zeta(\mathbf{z})$. Because the samples of $\mathbf{z}$ are replaced by a differentiable function $t(\boldsymbol{\beta}, \epsilon_z)$, then $\frac{\partial}{\partial \boldsymbol{\beta}} \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\beta}}(\mathbf{z})}[\zeta(\mathbf{z})] = \frac{\partial}{\partial \boldsymbol{\beta}} \mathbb{E}_{\epsilon_z \sim \text{Uniform}(0,1)}[\frac{\partial \zeta}{\partial t} \frac{\partial t}{\partial \boldsymbol{\beta}}]$, where $t(\cdot, \cdot)$ represents the whole transformation process in Prop. 1.

Recall that our objective is to approximate the chain of Bernoulli variables $p(\mathbf{z})$ with $q_{\boldsymbol{\beta}}(\mathbf{z})$. Property 2 shows why the proposed distribution is consistent with the chain of Bernoullis in essence, and provides an easy way to derive the evidence lower bound for variational inference. The proof for the two properties is in Appx. 1.1. This simple transformation of Gumbel softmax samples allows fast sampling of an ordered unit. Compared with $p(\mathbf{z})$, the complexity decreases from $\mathcal{O}(K)$ to $\mathcal{O}(1)$, as the sequential sampling of the Bernoulli chain is no longer necessary.

3. Bayesian Nested Neural Network

In this section, we present the Bayesian nested neural network based on the fundamental units proposed in Sec. 2.

3.1. Bayesian Inference and SGVB

Consider a dataset $\mathcal{D}$ constructed from N pairs of instances $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$. Our objective is to estimate the parameters $\mathbf{u}$ of a neural network $p(\mathbf{y}|\mathbf{x}, \mathbf{u})$ that predicts $\mathbf{y}$ given input $\mathbf{x}$ and parameters $\mathbf{u}$. In Bayesian learning, a prior $p(\mathbf{u})$ is placed over the parameters $\mathbf{u}$. After data $\mathcal{D}$ is observed, the prior distribution is transformed into a posterior distribution $p(\mathbf{u}|\mathcal{D})$.

For neural networks, computing the posterior distribution using the Bayes rule requires computing intractable integrals over $\mathbf{u}$. Thus, approximation techniques are required. One family of techniques is variational inference, with which the posterior $p(\mathbf{u}|\mathcal{D})$ is approximated by a parametric distribution $q_{\boldsymbol{\phi}}(\mathbf{u})$, where $\boldsymbol{\phi}$ are the variational parameters. $q_{\boldsymbol{\phi}}(\mathbf{u})$ is approximated by minimizing the Kullback-Leibler (KL) divergence with the true posterior, $\text{KL}[q_{\boldsymbol{\phi}}(\mathbf{u})||p(\mathbf{u}|\mathcal{D})]$, which is equivalent to maximizing the *evidence lower bound* (ELBO):

$$\mathcal{L}_{\boldsymbol{\phi}} = L_{\mathcal{D}}(\boldsymbol{\phi}) - \text{KL}[q_{\boldsymbol{\phi}}(\mathbf{u})||p(\mathbf{u})], \tag{4}$$

where the expected data log-likelihood is

$$L_{\mathcal{D}}(\boldsymbol{\phi}) = \sum_{i=1}^{N} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{u})}[\log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{u})]. \tag{5}$$

The integration $L_{\mathcal{D}}$ is not tractable for neural networks. An efficient method for gradient-based optimization of the variational bound is *stochastic gradient variational Bayes* (SGVB) [23, 22]. SGVB parameterizes the random parameters $\mathbf{u} \sim q_{\boldsymbol{\phi}}(\mathbf{u})$ as $\mathbf{u} = t(\boldsymbol{\epsilon}, \boldsymbol{\phi})$ where $t(\cdot)$ is a differentiable function and $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ is a noise variable with fixed parameters. With this parameterization, an unbiased differentiable minibatch-based Monte Carlo estimator of the expected data log-likelihood is obtained:

$$L_{\mathcal{D}}(\boldsymbol{\phi}) \simeq L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\phi}) = \frac{N}{M} \sum_{i=1}^{M} \log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{u} = t(\boldsymbol{\epsilon}, \boldsymbol{\phi})), \tag{6}$$

where $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^M$ is a minibatch of data with M random instances $(\mathbf{x}_i, \mathbf{y}_i) \sim \mathcal{D}$, and $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$.

3.2. Bayesian Nested Neural Network

In our model, the parameter $\mathbf{u} = (\mathbf{W}, \mathbf{z})$ consists of two parts: weight matrix $\mathbf{W}$ and ordering units $\mathbf{z}$. The ordering units order the network weights and generate sub-models that minimize the residual loss of a larger sub-model [37, 6]. We define the corresponding variational parameters $\boldsymbol{\phi} = (\boldsymbol{\theta}, \boldsymbol{\beta})$, where $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ are the variational parameters for the weights and ordering units respectively. We then have the following optimization objective,

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\beta}}^{\text{SGVB}} \simeq L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\theta}, \boldsymbol{\beta}) - \text{KL}[q_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{W}, \mathbf{z})||p(\mathbf{W}, \mathbf{z})], \tag{7}$$

$$L_{\mathcal{D}}^{\text{SGVB}}(\boldsymbol{\theta}, \boldsymbol{\beta}) = \frac{N}{M} \sum_{i=1}^{M} \log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{W} = t_w(\boldsymbol{\epsilon}_w, \boldsymbol{\theta}), \mathbf{z} = t_z(\boldsymbol{\epsilon}_z, \boldsymbol{\beta})), \tag{8}$$

where $\boldsymbol{\epsilon}_z$ and $\boldsymbol{\epsilon}_w$ are the random noise, and $t_w(\cdot)$ and $t_z(\cdot)$ are the differentiable functions that transform the noises to the probabilistic weights and ordered masks.

Next, we focus on an example of a fully-connected (FC) layer. Assume the FC layer in neural network takes in activations $\mathbf{H} \in \mathbb{R}^{M \times d}$ as the input, and outputs $\mathbf{F} = f(\mathbf{H}) = \mathbf{HW}$, where the weight matrix $\mathbf{W} \in \mathbb{R}^{d \times D}$, d and D are the input and output size, and M is the batch size. The elements are indexed as h_{mi}, f_{mj} and w_{ij} respectively. We omit the bias for simplicity, and our formulation can easily be extended to include the bias term. We have the ordering unit $\mathbf{z} \in \mathbb{R}^D$ with each element z_j applied on the column of $\mathbf{W}$, by which the columns of $\mathbf{W}$ are given different levels of importance. Note that $\mathbf{z}$ is flexible, and can be applied to $\mathbf{W}$ row-wise or element-wise as well.

The prior for $\mathbf{W}$ assumes each weight is independent, $p(\mathbf{W}) = \prod_{ij} p(w_{ij})$, where $i \in \{1, \ldots, d\}$ and $j \in \{1, \ldots, D\}$. We choose to place a mixture of two univariate variables as the prior over each element of the weight matrix w_{ij}. For example, if we use the univariate normal distribution, then each w_{ij} is a Gaussian mixture, where the 2 components are:

$$p(w_{ij}|z_j = 0) = \mathcal{N}(w_{ij}|\mu_{ij}^0, {\sigma_{ij}^0}^2), \tag{9}$$

$$p(w_{ij}|z_j = 1) = \mathcal{N}(w_{ij}|\mu_{ij}^1, {\sigma_{ij}^1}^2), \tag{10}$$

where $(\mu_{ij}^0, \sigma_{ij}^0)$ and $(\mu_{ij}^1, \sigma_{ij}^1)$ are the means and standard deviations for the two components. We fix $\mu_{ij}^0 = 0$ and σ_{ij}^0 to be a small value, resulting in a spike at zero for the component when $z_j = 0$. The variable z_j follows the chain of Bernoulli distributions proposed in (32). Using (2), the marginal distribution of w_{ij} is then $p(w_{ij}) = (1 - \prod_{k=1}^{i} \pi_k)\mathcal{N}(w_{ij}|\mu_{ij}^0, {\sigma_{ij}^0}^2) + (\prod_{k=1}^{i} \pi_k)\mathcal{N}(w_{ij}|\mu_{ij}^1, {\sigma_{ij}^1}^2)$.

To calculate the expected data log-likelihood, our *Downhill* distribution allows efficient sampling and differentiable transformation for the ordering units (Sec. 2.3). The reparameterization of weight distributions has been widely studied [22, 27, 23] to provide gradient estimate with low variance. Our framework is compatible with these techniques, which will be discussed in Sec. 3.4. An overview of sampling is shown in Fig. 1.

3.3. Posterior Approximation

Next, we introduce the computation of the KL divergence. We assume the posterior $q_{\boldsymbol{\theta}}(\mathbf{W})$ takes the same form as the prior, while $q_{\boldsymbol{\beta}}(\mathbf{z})$ takes the Downhill distribution $\mathbf{z} \sim \text{Downhill}(\boldsymbol{\beta}, \tau)$. We consider the case that $\tau \to 0$ for simplicity, while τ can be adjusted in the training process as annealing. For this layer, the KL divergence in (7) is

$$\begin{aligned} &\text{KL}[q_{\boldsymbol{\beta},\boldsymbol{\theta}}(\mathbf{W}, \mathbf{z})||p(\mathbf{W}, \mathbf{z})] \qquad (11) \\ &= \underbrace{\mathbb{E}_{q_{\boldsymbol{\beta}}(\mathbf{z})}[\log \frac{q_{\boldsymbol{\beta}}(\mathbf{z})}{p(\mathbf{z})}]}_{\Phi_1} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\beta}}(\mathbf{z})}\mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{W}|\mathbf{z})}[\log \frac{q_{\boldsymbol{\theta}}(\mathbf{W}|\mathbf{z})}{p(\mathbf{W}|\mathbf{z})}]}_{\Phi_2}. \end{aligned}$$

Term Φ_1 of (11) is

$$\Phi_1 = \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \log \frac{q_{\boldsymbol{\beta}}(\mathbf{z})}{p(\mathbf{z})} = \sum_{j=1}^{D} \text{KL}[q_{\boldsymbol{\beta}}(\mathbf{v}_i)||p(\mathbf{v}_i)],$$

where $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_D\}$ is the set of ordered masks. The number of components in the $\mathbf{z}$ space is reduced from D^2 to D, because there are only D possible ordered masks. By definition, the probabilities are

$$q_{\boldsymbol{\beta}}(\mathbf{v}_j) = \beta_j, \quad p(\mathbf{v}_j) = (1 - \pi_{j+1}) \prod_{k=1}^{j} \pi_k, \qquad (12)$$

where we define $\pi_{D+1} = 0$. The derivation of $p(\mathbf{v}_j)$ is included in the Appx. 1.2.

Define $\mathbf{w}_j = [w_{1j}, \ldots, w_{Dj}]$ as the j-th column of $\mathbf{W}$, and $q_{\boldsymbol{\theta}}(\mathbf{w}_j|z_j = k) = q_{\boldsymbol{\theta}}(\mathbf{w}_j|z_j^k)$ where $k \in \{0, 1\}$. The term Φ_2 of (11) is

$$\begin{aligned} \Phi_2 &= \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \sum_j \int_{\mathbf{w}_j} q_{\boldsymbol{\theta}}(\mathbf{w}_j|z_j^k) \log \frac{q_{\boldsymbol{\theta}}(\mathbf{w}_j|z_j^k)}{p(\mathbf{w}_j|z_j^k)} \\ &= \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \sum_i \sum_j \int_{w_{ij}} q_{\boldsymbol{\theta}}(w_{ij}|z_j^k) \log \frac{q_{\boldsymbol{\theta}}(w_{ij}|z_j^k)}{p(w_{ij}|z_j^k)} \\ &= \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \sum_{i,j} \text{KL}[q_{\boldsymbol{\theta}}(w_{ij}|z_j^k)||p(w_{ij}|z_j^k)]. \qquad (13) \end{aligned}$$

Note that the term inside the integration over w_{ij} is the KL divergence between the univariate conditional density in the prior and the posterior, with $z_j = 0$ or $z_j = 1$. Define $K_{ij}^k(\boldsymbol{\theta})$ as the KL of w_{ij} for component $k \in \{0, 1\}$. The term Φ_2 can then be re-organized as

$$\begin{aligned} \Phi_2 &= q(\mathbf{z} = \mathbf{v}_1)(\sum_i K_{i1}^1(\boldsymbol{\theta}) + \sum_{j=2}^{D} \sum_i K_{ij}^0(\boldsymbol{\theta})) \\ &+ q(\mathbf{z} = \mathbf{v}_2)(\sum_{j=1}^{2} \sum_i K_{ij}^1(\boldsymbol{\theta}) + \sum_{j=3}^{D} \sum_i K_{ij}^0(\boldsymbol{\theta})) \\ &+ \ldots \qquad (14) \end{aligned}$$

There are totally $D^2 d$ terms, which causes a large computation cost in every epoch. Consider the matrices $\mathbf{K}_{\boldsymbol{\theta}}^0 = [K_{ij}^0(\boldsymbol{\theta})]_{ij} \in \mathbb{R}^{d\times D}$ and $\mathbf{K}_{\boldsymbol{\theta}}^1 = [K_{ij}^1(\boldsymbol{\theta})]_{ij} \in \mathbb{R}^{d\times D}$, which are easily computed by applying the KL function element-wise. The term Φ_2 is then expressed as

$$\Phi_2 = \mathbf{e}^T \mathbf{K}_{\boldsymbol{\theta}}^0 (\mathbf{J} - \mathbf{J}_L)^T \boldsymbol{\beta} + \mathbf{e}^T \mathbf{K}_{\boldsymbol{\theta}}^1 \mathbf{J}_L^T \boldsymbol{\beta}, \qquad (15)$$

where $\mathbf{e}$ is a vector of 1s, $\mathbf{J}$ is a matrix of 1s and $\mathbf{J}_L$ is a lower triangular matrix with each element being 1. Then the calculation in (15) can be easily parallelize with modern computation library.

Ordered ℓ_0-Regularization. We show that, if given the spike-and-slab priors, our KL term in (11) has a nice interpretation as *a generalization of an ordered ℓ_0 regularization over the sub-networks*. The corresponding reduced objective for deterministic networks is

$$\min_{\boldsymbol{\theta},\boldsymbol{\beta}} \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\beta})} L_D(\boldsymbol{\theta}, \boldsymbol{\beta}) + \lambda \sum_j^D j\beta_j \qquad (16)$$

Note that larger sub-networks have greater penalization. The proof and more interpretations are in the Appx. 1.3.

3.4. Implementation

For efficient sampling of the weight distributions, we put multiplicative Gaussian noise $\eta_{ij} \sim \mathcal{N}(1, \alpha)$ on the weight w_{ij}, similar to [22, 31, 26]. We take w_{ij} for $z_j = 1$ as an example.

$$w_{ij} = \theta_{ij}\eta_{ij} = \theta_{ij}(1 + \sqrt{\alpha_{ij}}\epsilon_w), \ \epsilon_w \sim \mathcal{N}(0, 1), \qquad (17)$$

$$w_{ij} \sim \mathcal{N}(w_{ij}|\theta_{ij}, \alpha_{ij}\theta_{ij}^2). \qquad (18)$$

We also assume a log-uniform prior [22, 31, 26]. (10) becomes $p(\log|w_{ij}| \mid z_j = 1) = \text{const}$. With this prior, the negative KL term in (13) does not depend on the variational parameter θ_{ij}^1 [22], when the parameter α_{ij} is fixed,

$$\begin{aligned} &-\text{KL}[q(w_{ij}|\theta_{ij}^1, \alpha_{ij}, z_j = 1)||p(|w_{ij}| \,|z_j = 1)] \\ &= \frac{1}{2}\log\alpha_{ij} - \mathbb{E}_{\epsilon_w\sim\mathcal{N}(1,\alpha_{ij})} \log|\epsilon_w| + C, \qquad (19) \end{aligned}$$

where C is a constant.

As the second term in (19) cannot be computed analytically and should be estimated by sampling, Kingma *et al.* [22] propose to sample first and design a function to approximate it, but their approximation of $K_{ij}^1(\theta)$ does not

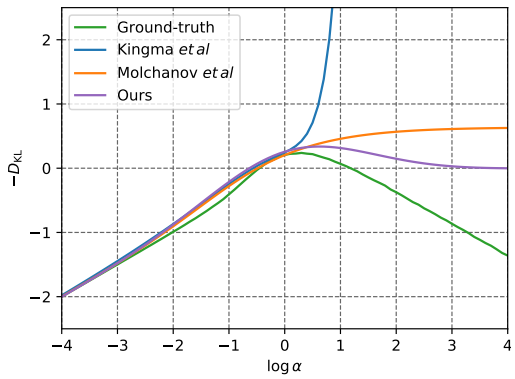

Figure 4: Approximation to (19). Our approximation allows $\alpha > 1$ (c.f., [22]) and does not push $\alpha \to 0$ to generate a collapsed model (c.f., [31]).

encourage $\alpha_{ij} > 1$ as the optimization is difficult. An $\alpha_{ij} \leq 1$ corresponds to a small variance, which is not flexible. Molchanov *et al*. [31] use a different parameterization that pushes $\alpha_{ij} \to \infty$, which means this w_{ij} can be discarded, as illustrated in Fig. 4. In our model, we want the order or sparsity of weights to be explicitly controlled by the ordering unit $\mathbf{z}$, otherwise the network would collapse to a single model rather than generate a nested set of sub-models. Thus, we propose another approximation to (19),

$$-\mathrm{KL}[q(w_{ij}|\theta^1_{ij}, \alpha_{ij}, z_j = 1)||p(|w_{ij}|\,|z_j = 1)] \quad (20)$$
$$\approx a_1 e^{-e^{a_4}\cdot(a_2+a_3*\log\alpha_{ij})^2} - 0.5\log(1+\alpha_{ij}^{-1}) + C,$$

where $a_1 = 0.7294$, $a_2 = -0.2041$, $a_3 = 0.3492$ and $a_4 = 0.5387$. We obtained these parameters by sampling from $\boldsymbol{\epsilon}_w$ to estimate (19) as the ground-truth and fit these parameters for 10^5 epochs. For fitting the curves, the input range is limited to $\log\alpha \in [-5, 0.5]$. As shown in Fig. 4, our parameterization allows $\alpha > 1$ and maximizing $-\mathrm{KL}$ does not push α to infinity (c.f. [22] and [31]), providing *more flexible* choices for the weight variance.

As the prior of the zero-component w_{ij} is assumed a spike at zero with a small constant variance, we let $q(w_{ij}|z_j = 0)$ be the same spike as (9) to save computation. Also, to speed up the sampling process in Fig. 1, we directly multiply the sampled mask with the output features of the layer. This saves the cost for sampling from $w_{ij}|z_j = 0$ and simplifies (15) to $\mathbf{e}^T\mathbf{K}^1_{\boldsymbol{\theta}}\mathbf{J}^T_L\boldsymbol{\beta}$. Using the notation in Sec. 3.2, the output of a fully connected layer is

$$f_{mj} = b_{mj}z_j, \quad b_{mj} \sim \mathcal{N}(\gamma_{mj}, \delta_{mj}), \quad (21)$$
$$\gamma_{mj} = \sum_{i=1}^{d} h_{mi}\theta_{ij}, \quad \delta_{mj} = \sum_{i=1}^{d} h^2_{mi}\alpha_{ij}\theta^2_{ij}. \quad (22)$$

The sampling process is similar to that of [22, 31, 26].

The Bayesian nested neural network can be easily extended to convolutional layers with the ordering applied to filter channels (see Appx. 2.1 for details).

4. Related work

In this section, we reviewed the deep nets with ℓ_0 regularization and nested nets, while the comparisons with Bayesian neural network are elaborated in Sec. 3.4.

ℓ_0 regularization. The Bernoulli-Gaussian linear model with independent Bernoulli variables is shown to be equivalent to ℓ_0 regularization [32]. Recent works [28, 48] investigate ℓ_0 norm for regularizing deep neural networks. [28] presents a general formulation of a ℓ_0-regularized learning objective for a single deterministic neural network,

$$\min_{\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\pi}}} \mathbb{E}_{q(\tilde{\mathbf{z}}|\tilde{\boldsymbol{\pi}})}[L_D(\tilde{\boldsymbol{\theta}})] + \lambda \sum_{j=1}^{|\tilde{\boldsymbol{\theta}}|} \tilde{\pi}_j \quad (23)$$

where the variable $\tilde{\mathbf{z}}$ is a binary gate with parameter $\tilde{\pi}_j$ for each network node $\tilde{\theta}_j$, and L_D is the loss. It was shown that ℓ_0 regularization over the weights is a special case of an ELBO over parameters with spike-and-slab priors. These works present the *uniform* ℓ_0 regularization as the coefficient λ is a constant over the weights. It is interesting that our ELBO (7) can be viewed as a generalization of a new training objective of deterministic networks, which includes a weighted penalization over the choices of sub-networks, interpretable as *an ordered ℓ_0 regularization* (16).

Nested neural networks. Nested nets have been explored in recent years, for its portability in neural network (NN) deployment on different platforms. [20] proposes a network-in-network structure for a nested net. which consists of internal networks from the core level to the full level. [50, 49] propose slimmable NN that trains a network that samples multiple sub-networks of different channel numbers (widths) simultaneously, where the weights are shared among different widths. The network needs to switch between different batch normalization parameters that correspond to different widths. To alleviate the interference in optimizing channels in slimmable NN, [4] proposes a once-for-all network that is elastic in kernel size, network depth and width, by shrinking the network progressively during training. [6] proposes using *nested dropout* to train a fully nested neural network, which generates more sub-networks in nodes, including weights, channels, paths, and layers. However, *none* of the previous works consider learned importance over the nodes and the predictive uncertainty. Our work provides a well-calibrated uncertainty and the learned importance, with a full Bayesian treatment of nested nets.

5. Experiments

We next present experiments using our Bayesian nested neural network. The experiments include two main tasks: image classification and semantic segmentation.

5.1. Image Classification

Datasets and setup. The image classification experiments are conducted on Cifar10 and Tiny Imagenet (see Appx. 3.4 for results on Cifar100). The tested NN models are VGG11 with batch normalization layers [41], ResNeXt-Cifar model from [46], and MobileNetV2 [40]. To train

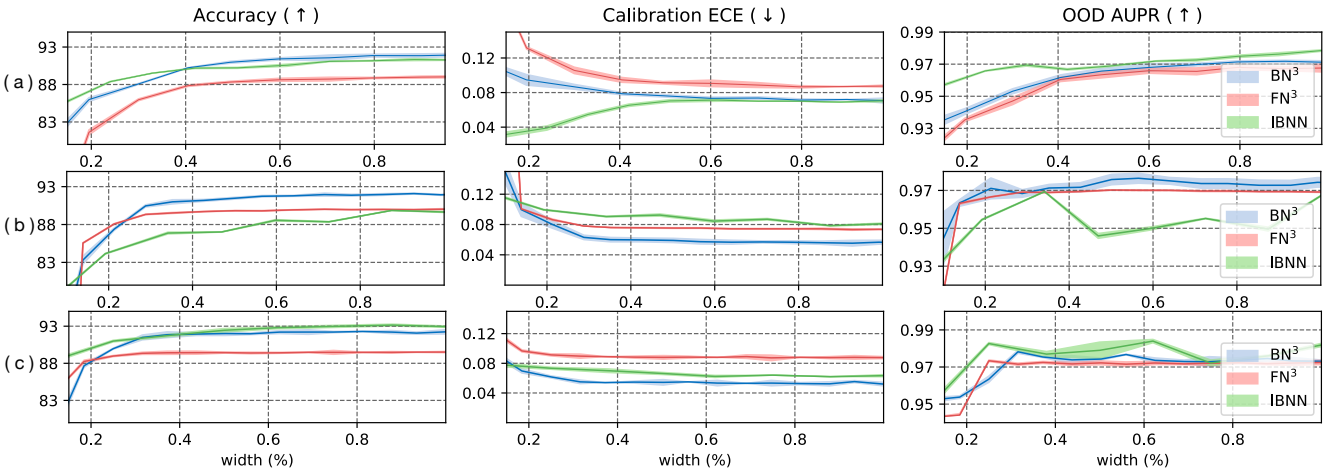

Figure 5: Results on Cifar10 for (a) VGG11, (b) MobileNetv2, and (c) ResNeXt-Cifar. Each curve plots performance versus the network width. The solid line indicate the mean and the shaded area indicates two standard deviations.

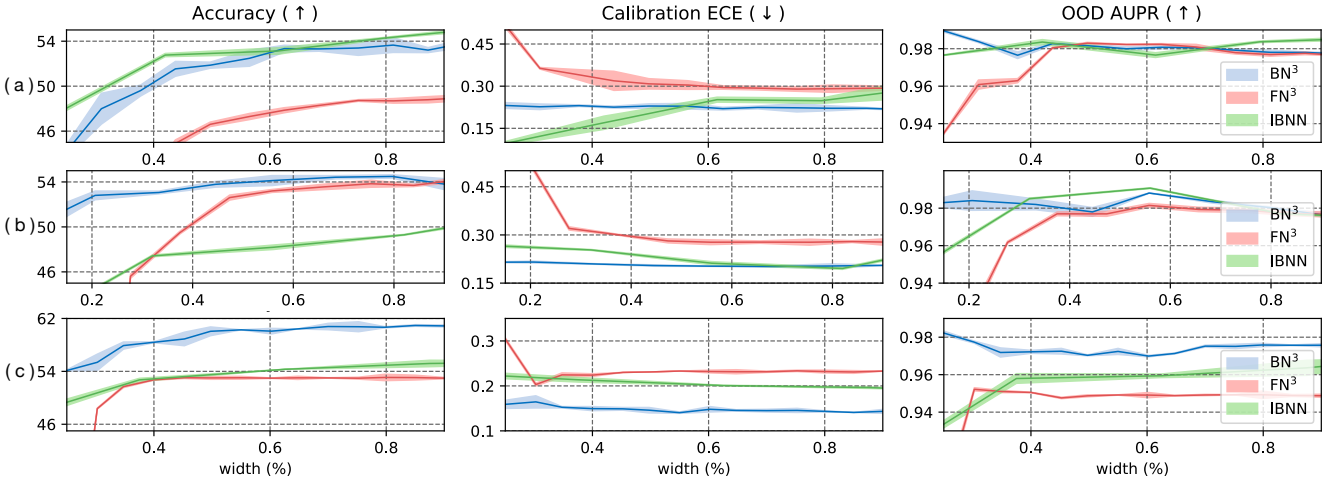

Figure 6: Results on Tiny ImageNet for (a) VGG11, (b) MobileNetv2, (c) ResNeXt-Cifar.

the proposed Bayesian nested neural network (denoted as BN^3), we use the cross-entropy loss for the expected log-likelihood in (8). The computation of the KL term follows Sec. 3.4. For ordering the nodes, in every layer, we assign each dimension of the prior (Bernoulli chain) and posterior (Downhill variable) of the ordering unit to a group of weights. Thus, the layer width is controlled by the ordering unit. We set the number of groups to 32 for VGG11 and ResNext-Cifar, and to 16 for MobileNetV2. We compare our BN^3 with the fully nested neural network (FN^3) [6], since it can be seen as an extension of slimmable NN [50, 49] to fine-grained nodes. We also compare with the Bayesian NN with variational Gaussian dropout [22], where we train a set of independent Bayesian NNs (IBNN) for different fixed widths. Conceptually, the performance of IBNN, which trains separate sub-networks, is the ideal target for BN^3, which uses nested sub-networks.

During testing time, we generate fixed width masks for BN^3 and FN^3 as in [6]. For fairness, we do not perform local search of optimal width like in FN^3, but directly truncating the widths. We re-scale the node output by the probability that a node is kept (see Appx. 2.2). The batch normalization statistics are then re-collected for 1 epoch with the training data (using fewer data is also feasible as shown in Appx. 3.2). The number of samples used in testing $\text{BN}^3$and IBNN is 6. The detailed hyper-parameter settings for training and testing are in Appx. 3.1.

Evaluation metrics. For the evaluation, we test accuracy, uncertainty calibration, and out-of-domain (OOD) detection. Calibration performance is measured with the expected calibration error [12] (ECE), which is the expected difference between the average confidence and accuracy. OOD performance is measured with the area under the precision-recall curve (AUPR) [3, 15, 25] (see Appx. 3.3 for AUROC curves). If we take the OOD class as positive, precision is the fraction of detected OOD data that

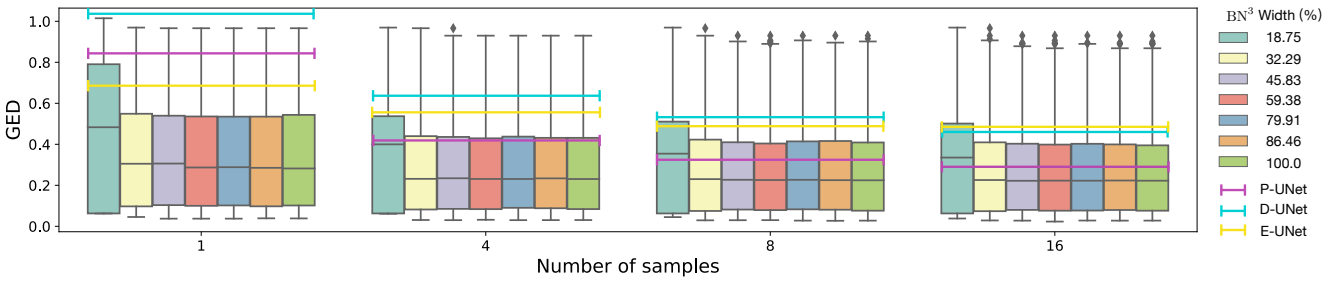

Figure 7: Evaluation on semantic segmentation using generalized energy distance (↓) with different numbers of posterior samples. Each box-plot shows the GED of all data of BN^3 for one network width (%). The black horizontal line in the box plot represents the mean. The bold horizontal lines represent the averaged results for comparison methods, which use the full-width network.

are true OOD, while recall is the fraction of true OOD data that are successfully detected. Note that a better model will have higher accuracy and OOD AUPR, and lower calibration ECE. As the sampling and collection of batch-norm statistics are stochastic, we repeat each trial 3 times and report the average results.

Results. The results are presented in Figs. 5 and 6. First, looking at performance versus width, BN^3 exhibits the well-behaved property of sub-networks, where the performance increases (accuracy and AUPR increase, ECE decreases) or is stable as the width increases. This demonstrates that the variational ordering unit successfully orders the information in each layer.

Despite learning nested sub-networks, in general, BN^3 has similar performance as IBNN (which separately learns sub-networks) for all models and datasets, with the following exceptions. For MobileNetV2 on both datasets, BN^3 outperforms IBNN in all metrics, as IBNN fails to perform well in prediction and uncertainty (outperformed by FN^3 too). For VGG11 on both datasets, IBNN tends to have lower ECE with smaller widths, showing its advantage in providing uncertainty for small and simple models. However, IBNN has larger ECE when the model size is large, e.g., BN^3 has lower ECE than IBNN with the ResNeXt model. Finally, BN^3 outperforms IBNN by a large margin for ResNeXt on Tiny ImageNet, which we attribute to its ability to prune the complex architecture via learning ordered structures (Sec. 2.3) and the ordered ℓ-0 regularization effect (Sec. 3.3), which are absent in IBNN.

Comparing the two nested models, BN^3 outperforms FN^3 in all metrics, which shows the advantage of learning the nested dropout rate for each node.

5.2. Lung Abnormalities Segmentation

Dataset and Setup. The semantic segmentation experiments are conducted on the LIDC-IDRI [5] dataset, which contains 1,018 CT scans from 1,010 lung patients with manual lesion segmentation from four experts. This dataset is uncertainty-critical as it contains typical ambiguities in labels that appear in medical applications. We follow [24] to process the data, resulting in 12,870 images in total. We adopt the *generalized energy distance* (GED) [2, 39, 43, 24] as the evaluation metric, with $\delta(\cdot,\cdot) = 1 - \text{IoU}(\cdot,\cdot)$ as the distance function. GED measures the distance between the output distributions rather than single deterministic predictions. For BN^3, it measure the probabilistic distances between the induced distribution from model posterior given a fixed width, and the noisy labels from four experts. We use a standard U-Net [38] for BN^3 and the number of groups is 32. We compare with Probabilistic U-Net (P-UNet) [24], a deep ensemble of U-Net (E-UNet), and Dropout U-Net (D-UNet) [18]. Their results are the average results from [24] with the full U-Net.

Results. The results are presented in Fig. 7. We observe that BN^3 outperforms the existing methods in most of the cases, with the difference more obvious when there are fewer posterior samples. The performance of BN^3 stabilizes after width of 32.29%. This indicates BN^3 learns a compact and effective structure compared with other methods, in terms of capturing ambiguities in the labels.

When there are more posterior samples (8 and 16), probabilistic U-Net has better performance than the BN^3 with the smallest width ($\frac{6}{32}$ channels are preserved). This means with more posterior samples, the probabilistic U-Net can depict the latent structure better, but uses a full-width model. Increasing the width to 32.29%, BN^3 then achieves better performance.

6. Conclusion

In this paper, we propose a Bayesian nested neural network, which is based on a novel variational ordering unit that explicitly models the weight importance via the Downhill random variable. From our model, the weight importance can be learned from data, rather than hand-tuned as with previous methods. Experiments show that this framework can improve both accuracy and calibrated predictive uncertainty. Future work will study the variational ordering unit in language modeling, sequential data, or generative models where the order is important, e.g., [35]. The Downhill random variable is a well-suited hidden variable for such applications.

References

[1] Artur Bekasov and Iain Murray. Ordering dimensions with nested dropout normalizing flows. *arXiv preprint arXiv:2006.08777*, 2020. 1

[2] Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The cramer distance as a solution to biased wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017. 8

[3] Kendrick Boyd, Kevin H Eng, and C David Page. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 451–466. Springer, 2013. 7

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 1, 6

[5] Kenneth Clark, Bruce Vendt, Kirk Smith, John Freymann, Justin Kirby, Paul Koppel, Stephen Moore, Stanley Phillips, David Maffitt, Michael Pringle, et al. The cancer imaging archive (tcia): maintaining and operating a public information repository. *Journal of digital imaging*, 26(6):1045–1057, 2013. 8

[6] Yufei Cui, Ziquan Liu, Wuguannan Yao, Qiao Li, Antoni B. Chan, Tei-wei Kuo, and Chun Jason Xue. Fully nested neural network for adaptive compression and quantization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2080–2087. International Joint Conferences on Artificial Intelligence Organization, 7 2020. 1, 2, 4, 6, 7

[7] Chelsea Finn, Lisa Anne Hendricks, and Trevor Darrell. Learning compact convolutional neural networks with nested dropout. *arXiv preprint arXiv:1412.7155*, 2014. 1

[8] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016. 2

[9] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016. 1, 2

[10] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in neural information processing systems*, pages 3581–3590, 2017. 2, 3

[11] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1948. 11

[12] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599*, 2017. 1, 7

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[14] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017. 14

[15] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016. 7

[16] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016. 14

[17] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 1, 3, 11

[18] 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. 1, 8

[19] Mohammad Emtiyaz Khan and Didrik Nielsen. Fast yet simple natural-gradient descent for variational inference in complex models. In *2018 International Symposium on Information Theory and Its Applications (ISITA)*, pages 31–35. IEEE, 2018. 14

[20] Eunwoo Kim, Chanho Ahn, and Songhwai Oh. Nestednet: Learning nested sparse structures in deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8669–8678, 2018. 1, 6

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[22] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015. 4, 5, 6, 7, 13

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 4, 5

[24] Simon Kohl, Bernardino Romera-Paredes, Clemens Meyer, Jeffrey De Fauw, Joseph R Ledsam, Klaus Maier-Hein, SM Eslami, Danilo Jimenez Rezende, and Olaf Ronneberger. A probabilistic u-net for segmentation of ambiguous images. *Advances in neural information processing systems*, 31:6965–6975, 2018. 1, 8

[25] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017. 7

[26] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in neural information processing systems*, pages 3288–3298, 2017. 5, 6

[27] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017. 5

[28] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018. 6, 12

[29] 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. 1, 3, 11

[30] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Advances in Neural Information Processing Systems*, pages 3086–3094, 2014. 11

[31] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017. 5, 6, 13

[32] Kevin P Murphy. *Machine learning: a probabilistic perspective*. 2012. 6

[33] Min-hwan Oh, Peder A Olsen, and Karthikeyan Natesan Ramamurthy. Crowd counting with decomposed uncertainty. *arXiv preprint arXiv:1903.07427*, 2019. 1

[34] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. Parallel training of dnns with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*, 2014. 14

[35] Piyush Rai, Changwei Hu, Ricardo Henao, and Lawrence Carin. Large-scale bayesian multi-label learning via topic-based label embeddings. In *Advances in Neural Information Processing Systems*, pages 3222–3230, 2015. 8

[36] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 925–938, 2019. 1

[37] Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pages 1746–1754, 2014. 1, 2, 4

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 8

[39] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving gans using optimal transport. *arXiv preprint arXiv:1803.05573*, 2018. 8

[40] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 6

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 13

[43] Gábor J Székely and Maria L Rizzo. Energy statistics: A class of statistics based on distances. *Journal of statistical planning and inference*, 143(8):1249–1272, 2013. 8

[44] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 1

[45] Jia Wan and Antoni Chan. Modeling noisy annotations for crowd counting. In *Advances in Neural Information Processing Systems*, 2020. 1

[46] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 6

[47] Shichao Xu, Yixuan Wang, Yanzhi Wang, Zheng O'Neill, and Qi Zhu. One for many: Transfer learning for building hvac control. *arXiv preprint arXiv:2008.03625*, 2020. 1

[48] Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*, 2019. 6

[49] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1803–1811, 2019. 1, 6, 7

[50] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018. 1, 6, 7

7. Appendix - Derivation and Proofs

7.1. Derivation of Properties

Property 1. If $\mathbf{c} \sim \text{Gumbel\_softmax}(\tau, \beta, \epsilon_z)$[2], then $z_i = 1 - \text{cumsum}'_i(\mathbf{c})$, where $\mathbf{e}$ is a K-dimensional vector of ones, and $\text{cumsum}'_i(\mathbf{c}) = \sum_{j=0}^{i-1} c_j$. $c_0 \coloneqq 1$. ϵ_z is a standard uniform variable.

We show that using the sampling process in Property 1 recovers produce the Downhill random variable. We assume $\mathbf{c}$ follows a Gumbel softmax distribution [11, 30] which has the following form.

$$\begin{aligned} &p(c_1, \dots, c_K) \\ =&\Gamma(K)\tau^{K-1}(\sum_{i=1}^{K} \pi_i/c^{\tau})^{-K} \prod_{i=1}^{K}(\pi_i/c^{\tau+1}) \end{aligned} \tag{25}$$

We apply the transformation $T_i(\cdot) = \mathbf{e}_i - \text{cumsum}'_i(\cdot)$ to the variable $\mathbf{c}$. $\mathbf{z} = T(\mathbf{c}) = \mathbf{e} - \text{cumsum}'_i(\mathbf{c})$

To obtain the distribution of $p(\mathbf{z})$, we apply the change of variables formula on $\mathbf{c}$.

$$p(\mathbf{z}) = p(T^{-1}(\mathbf{z}))|\det(\partial \frac{T^{-1}(\mathbf{z})}{\partial \mathbf{z}})| \tag{26}$$

$$p(z_{1:K}) = p(T^{-1}(z_{1:K}))|\det(\frac{\partial T^{-1}(z_{1:K})}{\partial z_{1:K}})| \tag{27}$$

From the definition of $T(\cdot)$, we can obtain $T_i^{-1}(\mathbf{z}) = z_{i-1} - z_i$. The Jacobian

$$\frac{\partial T^{-1}(z_{1:K})}{\partial z_{1:K}} = \begin{bmatrix} -1 & 0 & \dots & 0 & 0 \\ 1 & -1 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -1 \end{bmatrix} \tag{28}$$

Thus, $|\det(\frac{\partial T^{-1}(z_{1:K})}{\partial z_{1:K}})| = 1$.

$$\begin{aligned} &p(z_{1:K}) = p(T_{1:K}^{-1}(\mathbf{z})) \\ =&\Gamma(K)\tau^{K-1}(\sum_{i=1}^{K} \frac{\pi_i}{(z_{i-1} - z_i)^{\tau}})^{-K} \prod_{i=1}^{K}(\frac{\pi_i}{(z_{i-1} - z_i)^{\tau+1}}) \end{aligned} \tag{29}$$

Property 2. When $\tau \to 0$, sampling from the Downhill distribution reduces to discrete sampling, where the sample space is the set of ordered mask vectors $\mathcal{V}$. The approximation of the Downhill distribution to the Bernoulli chain can be calculated in closed-form.

As shown in [29, 17], when $\tau \to 0$, the Gumbel softmax transformation corresponds to an argmax operation that generates an one-hot variable:

$$\mathbf{c} = \text{one\_hot}(\text{argmax}_i(g_i + \log \beta_i)), \tag{30}$$

where the relative order is preserved.

Say a sample $\mathbf{c}^* \sim \text{Gumbel\_softmax}(\boldsymbol{\beta}, \tau \to 0)$, with b-th entry being one and the rest entries being 0. The defined transformation generates $\text{cumsum}'(\mathbf{c}^*) = [\underbrace{0, \dots, 0}_{b} \underbrace{1, \dots, 1}_{K-b}]$. Thus, $\mathbf{z}^* = \mathbf{e} - \text{cumsum}'(\mathbf{c}^*) = [\underbrace{1, \dots, 1}_{b} \underbrace{0, \dots, 0}_{K-b}]$. It is easy to see the transformation $t'(\cdot)$ is surjective function that $t' : \{\text{one\_hot}(i)\}_{i=1}^{K} \to \mathcal{V}, t'(\mathbf{c}) = \mathbf{e} - \text{cumsum}'(\mathbf{c})$. $\mathcal{V}$ is exactly the set of ordered mask defined in Sec. 2.2.

Thus, we can calculate the approximation of Downhill variable to the Bernoulli chain,

$$\text{KL}[q(\mathbf{z})||p(\mathbf{z})] = \sum_{j=1}^{K} q(\mathbf{v}_j) \log \frac{q(\mathbf{v}_j)}{p(\mathbf{v}_j)}, \tag{31}$$

where $q(\mathbf{v}_j) = \beta_j, p(\mathbf{v}_j) = (1 - \pi_{j+1}) \prod_{k=1}^{j} \pi_k$ (See Appx 1.2). The KL divergence in (31) minimized to 0 when $\beta_j = (1 - \pi_{j+1}) \prod_{k=1}^{j}, \forall j \in [1, \dots, K]$.

[2]For Gumbel-softmax sampling, we first draw $g_1 \dots g_K$ from $\text{Gumbel}(0, 1)$, then calculate $c_i = \text{softmax}(\frac{\log(\beta_i) + g_i}{\tau})$. The samples of $\text{Gumbel}(0, 1)$ can be obtained by first drawing $\epsilon_z \sim \text{Uniform}(0, 1)$ then computing $g = -\log(-\log(\epsilon_z))$.

7.2. Probability of ordered masks

Recall the formulation of Bernoulli chain:

$$\begin{aligned} &p(z_1 = 1) = \pi_1, && p(z_1 = 0) = 1 - \pi_1, \\ &p(z_i = 1|z_{i-1} = 1) = \pi_i, && p(z_i = 0|z_{i-1} = 1) = 1 - \pi_i, \\ &p(z_i = 1|z_{i-1} = 0) = 0, && p(z_i = 0|z_{i-1} = 0) = 1, \end{aligned} \tag{32}$$

It is observed, there is a chance $z_i = 1$ only when $z_{i-1} = 1$, and $z_{>i} = 0$ if $z_i = 0$. Thus,

$$p(\mathbf{z} = \mathbf{v}_j) = (1 - \pi_{j+1}) \prod_{k=1}^{j} \pi_k, \tag{33}$$

where $j + 1$ is the index of first zero. And we define $\pi_{K+1} = 0$ as $p(\mathbf{z} = \mathbf{v}_K) = \sum_{k=1}^{K} \pi_k$, which means all nodes are remained.

7.3. ℓ-0 regularization

We consider the case when prior over each weight is a spike-and-slap distribution, i.e., $p(w_{ij}|z_j = 0) = \delta(w_{ij})$ and $p(w_{ij}|z_j = 1) = \mathcal{N}(w_{ij}|0, 1)$, using the notation in Sec. 3.3. The posterior is also in this form. The derivations of KL term in (11-15) remain unchanged as it make nothing but mean-field assumption on the weight prior. With Φ_1 and Φ_2 (15), the objective (7) can be re-organized as

$$\begin{aligned} &\mathcal{L}^{\text{SGVB}}_{\boldsymbol{\theta},\boldsymbol{\beta}} \\ \simeq &L^{\text{SGVB}}_{\mathcal{D}}(\boldsymbol{\theta}, \boldsymbol{\beta}) - \sum_{j=1}^{D} \text{KL}[q_{\boldsymbol{\beta}}(\mathbf{v}_i)||p(\mathbf{v}_i)] \\ &- \mathbf{e}^T \mathbf{K}^0_{\boldsymbol{\theta}} (\mathbf{J} - \mathbf{J}_L)^T \boldsymbol{\beta} - \mathbf{e}^T \mathbf{K}^1_{\boldsymbol{\theta}} \mathbf{J}^T_L \boldsymbol{\beta} \\ = &L^{\text{SGVB}}_{\mathcal{D}}(\boldsymbol{\theta}, \boldsymbol{\beta}) - \sum_{j=1}^{D} \text{KL}[q_{\boldsymbol{\beta}}(\mathbf{v}_i)||p(\mathbf{v}_i)] \\ &- \mathbf{e}^T \mathbf{K}^1_{\boldsymbol{\theta}} \mathbf{J}^T_L \boldsymbol{\beta}, \end{aligned} \tag{34}$$

as $\text{KL}[q(w_{ij}|z_j = 0)||p(w_{ij}|z_j = 0)] = 0$. We assume $\text{KL}[q(w_{ij}|z_j = 1)||p(w_{ij}|z_j = 1)] = \chi$ as in [28]. It means transforming $p(w_{ij}|z_j = 1)$ to $q(w_{ij}|z_j = 1)$ requires χ nats. Thus, $\mathbf{K}^1_{\boldsymbol{\theta}} = [\chi]_{d \times D}$. The last term is then simplified to

$$-\chi d \sum_{j=1}^{D} j \beta_j \tag{35}$$

Then,

$$\begin{aligned} &\mathcal{L}^{\text{SGVB}}_{\boldsymbol{\theta},\boldsymbol{\beta}} \\ = &L^{\text{SGVB}}_{\mathcal{D}}(\boldsymbol{\theta}, \boldsymbol{\beta}) - \sum_{j=1}^{D} \text{KL}[q_{\boldsymbol{\beta}}(\mathbf{v}_i)||p(\mathbf{v}_i)] - \chi d \sum_{j=1}^{D} j \beta_j, \end{aligned} \tag{36}$$

$$\leq L^{\text{SGVB}}_{\mathcal{D}}(\boldsymbol{\theta}, \boldsymbol{\beta}) - \chi d \sum_{j=1}^{D} j \beta_j \tag{37}$$

where the line 2-3 is because KL is positive. Let $\lambda = \chi d$. Then, maximizing the evidence lower bound presents the same objective in (16). This objective assigns greater penalization to the larger sub-networks with more redundant nodes. To compare with (23) [28] that uses have a constant coefficient over the probabilities, our reduced formulation provides an *ordered ℓ-0 regularization* instead of a *uniform ℓ-0 regularization*.

Note that (36) ignores the weight uncertainty compared with (7). (37) further ignores the uncertainty over the ordered mask, reduced to a deterministic formulation for a *nested neural network with learned weight importance*. The network used in this paper is (7) with weight uncertainty considered, where the detailed discussion for weight distributions.

8. Appendix - Implementation

8.1. Extension to Convolutional Layer

We consider a convolutional layer takes in a single tensor $\mathbf{H}_m^{H\times W\times C}$ as input, where m is the index of the batch, H, W and C are the dimensions of feature map. The layer has D filters aggregated as $\mathbf{w}^{D\times H'\times W'\times C}$ and outputs a matrix $\mathbf{F}_{mj}^{\bar{H}\times\bar{W}}$. In the paper, we consider the ordered masks applied over the output channels and each filter corresponds to a dimension in $\mathbf{z}$. As shown in [22, 31], the local reparameterization trick can be applied, due to the linearity of the convolutional layer.

$$f_{mj} = b_{mj} z_j^*, \quad \text{vec}(b_{mj}) \sim \mathcal{N}(\gamma_{mj}, \delta_{mj}) \tag{38}$$
$$\gamma_{mj} = \text{vec}(\mathbf{H}_m * \mathbf{w}), \quad \delta_{mj} = \text{diag}(\text{vec}(\mathbf{H}_m^2 * \sigma_j^2))$$

where z_j^* is the j-th dimension of the sampled ordered mask $\mathbf{z}^* = \mathbf{v}^* \sim q_{\boldsymbol{\beta}}(\mathbf{z})$.

To calculate the KL term (11), the only modification is to let the first summation be over the height, width and input channels in (13).

$$\Phi_2 = \sum_{\mathbf{z}\in\mathcal{V}} q_{\boldsymbol{\beta}}(\mathbf{z}) \sum_{i}^{H'\times W'\times C} \sum_{j}^{D} \int_{w_{ij}} q_{\boldsymbol{\theta}}(w_{ij}|z_j^k) \log \frac{q_{\boldsymbol{\theta}}(w_{ij}|z_j^k)}{p(w_{ij}|z_j^k)} \tag{39}$$

8.2. Re-scale weights for testing

During training, the network drops nodes with the variational nested dropout. In training, the network fixes width of each layer and no dropout operation is adopted. To make the expectation consistent over training and testing [42], we re-scale the weights according to the probability to keep a node.

$$\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z}),\mathbf{x}\sim\mathcal{D}_{\text{tr}}}[\mathbf{F}|\mathbf{x},\mathbf{z}] \approx \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{te}}}[\mathbf{F}|\mathbf{x},\mathbf{z}=\bar{\mathbf{v}}], \tag{40}$$

where $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{te}}$ are the splits of training set and testing set, and $\bar{\mathbf{v}}$ is the user-specified width according to the real demand during testing time.

We take the fully-connected layer as an example. For simplicity, we treat w_{ij} as deterministic here.

$$\begin{aligned}
&\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}[f_{mj}] \\
=&\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}[z_j \sum_{i=1}^{d} h_{mi}\theta_{ij}] \\
=&\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}[z_j] \sum_{i=1}^{d} h_{mi}\theta_{ij}
\end{aligned} \tag{41}$$

Note that, different from the probability $q_\beta(\mathbf{v}_j) = \beta_j$, $\mathbb{E}_{\mathbf{z}\sim q_\beta(\mathbf{z})}[z_j]$ is the probability that the j-th node being kept.

With a well-trained layer in Bayesian nested neural network, we have the learned importance $\boldsymbol{\beta} = [\beta_j]_j$. Assume that the $\boldsymbol{\beta}$ is also generated by a chain of hidden Bernoulli variables follows (1) with the parameters $\boldsymbol{\mu} = [\mu_j]_j$, with $\mu_1 \coloneqq 1$ and $\mu_j = q(z_j = 1|z_{j-1} = 1)$. We are interested in the marginal distribution $p(z_j = 1) = \prod_{k=1}^{j} \mu_k$ but we only have β_j's.

$$\begin{aligned}
\beta_1 &= (1-\mu_2)\mu_1 = 1-\mu_2 \\
\beta_2 &= (1-\mu_3)\mu_2\mu_1 \\
&\dots
\end{aligned} \tag{42}$$

Solving each equation sequentially, we obtain

$$\begin{aligned}
&p(z_1 = 1) = 1, \\
&p(z_2 = 1) = 1-\beta_1, \\
&p(z_3 = 1) = 1-\beta_1-\beta_2, \\
&\dots
\end{aligned} \tag{43}$$

(41) becomes

$$(1 - \sum_{k=1}^{j-1} \beta_k) \sum_{i=1}^{d} h_{mi}\theta_{ij}, \tag{44}$$

where we can define $\beta_0 = 0$. Then, the *scaling factor* is $1 - \sum_{k=1}^{j-1} \beta_k$ for each w_{ij}.

Another way is to optimize the conditional probabilities $[\mu_j]_j$ instead of β_j, with β_j in previous derivation replaced

by $(1-\mu_{j+1})\mu_j$[3]. The scaling factor is then $\prod_{k=1}^{j}\mu_k$ for each w_{ij}. Also, for simplicity, one can optimize $\bar{\mu}_k$ where $\mu_k = \text{sigmoid}(\bar{\mu}_k)$.

9. Appendix - Experiments

9.1. Experimental setups

We implement FN^3, individual Bayesian neural networks (IBNN) and the proposed Bayesian Nested Neural Network (BN^3) with PyTorch framework. We use the cross-entropy loss for negative expected log-likelihood. For balancing the regularization and likelihood, we add a scaling factor κ for the KL term, which is a common trick in Bayesian learning [16].

Cifar10/Cifar100. For data augmentation, we use random cropping with padding beforehand, and random flipping the image horizontally.

VGG11: We train BN^3-VGG11 with natural gradient descent[4] (NGD) [34], as it was shown to make the Bayesian neural network converge faster [19]. The network is trained for 600 epochs with an initial learning rate 0.1 and momentum 0.9. The learning rate is scaled by a factor 0.1 every 150 epochs. κ is set to 10^{-5}. For training the network, we use VGG11 with $1.5\times$ number of channels and truncate the $2/3$ part with higher importance for testing. We add one dense layer after the stack of convolutional layers. The first feature extraction layer and the last two dense layers for classification are variational Bayes layer without nested dropout, with our parameterization proposed in Sec. 3.4. For the convolutional layer, we divide the convolutional filters into 32 groups for group sparsity. 30 groups are applied nested dropout while the rest 2 groups are for extracting the basic features. The $\log\alpha_{ij}$ is initialized to -8 for the first layer and -1 for the rest layers. The $[\bar{\mu}_j]_j$ are all initialized to 3. We train IBNN-VGG11 with NGD for 240 epochs, with an initial learning rate 0.1 and scaled by 0.3 every 40 epochs. Every individual Bayes NN is fixed at some width between the fraction 0 and 1. We train FN^3-VGG11 with SGD and momentum 0.9, as SGD performs better in training FN^3-VGG11. Other setups are similar to BN^3-VGG11.

MobileNetV2: We train BN^3-MobileNetV2 with a similar setup as BN^3-VGG11, except the followings. For inverted residual block, we apply nested dropout to the middle depth-wise convolutional layer. Because it already sparsifies the convolution filters in the previous point-wise convolution layer, and channels in the following point-wise convolution layer [14]. Introducing more nested dropout units would cause extra and irregular sparsification which deteriorates the performance. We use a normal-size MobileNetV2 and divide the weights into 16 groups. One group is fixed for base feature extraction. The experimental setups for IBNN-MobileNetV2 and FN^3-MobileNetV2 follow that on VGG11.

ResNeXt-Cifar: The setups for ResNeXt-Cifar are similar to that of MobileNetV2, while the number of groups is 32.

Tiny-ImageNet. For data augmentation, we use random cropping with padding beforehand, random rotation of 20 degree and random flipping the image horizontally. All images are finally cropped to 64×64 and all networks are trained from scratch.

VGG11: To increase the capacity, we take VGG11 with $1.5\times$ number of channels as the base network. The network is trained with NGD for 300 epochs with an initial learning rate 0.1. The learning rate is scaled by 0.3 every 25 epochs. κ is set to 10^{-6}. The weights are divided into 32 groups and 8 groups are fixed for base feature extraction. *MobileNetV2*: We train a MobileNetV2 with $1.5\times$ number of channels, and take the $2/3$ part with higher importance as the base network for testing. The network is trained with NGD for 300 epochs with an initial learning rate 0.1. The learning rate is scaled by 0.3 every 40 epochs. The weights are divided into 16 groups and 1 groups are fixed for base feature extraction. *ResNeXt-Cifar*: We train a ResNeXt-Cifar with normal size. The weights are divided into 32 groups and 8 groups are fixed for base feature extraction. The network is trained with SGD for 300 epochs with an initial learning rate 0.1. The learning rate is scaled by 0.3 every 30 epochs.

The rest setups are similar to that on Cifar10/Cifar100.

Lung Abnormalities Segmentation. The network uses a U-Net shape architecture with layers 32-64-128-192 for the encoder (two layers fewer than the standard U-Net). The optimizer is Adam with initial learning rate 10^{-4} decayed by 0.1 every 60 epochs. The channels are divided into 32 groups and 6 groups are fixed for base feature extraction.

[3] We use this parameterization in our implementation, while we use β_j in most of our derivation for simplicity in writing.

[4] The PyTorch implementation is from https://github.com/YiwenShaoStephen/NGD-SGD.

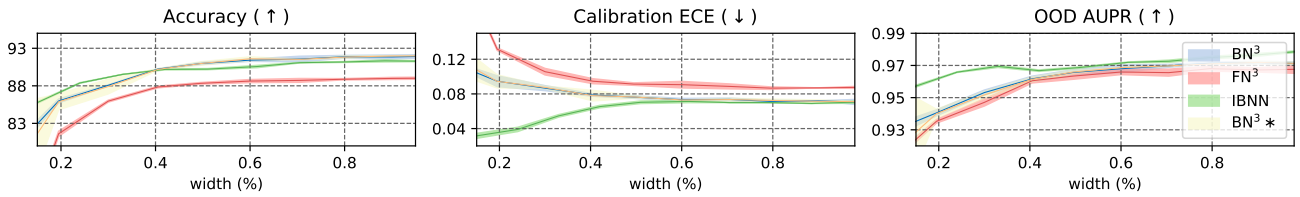

Figure 8: Performance of VGG11 on Cifar10 with less data for BN statistics collection.

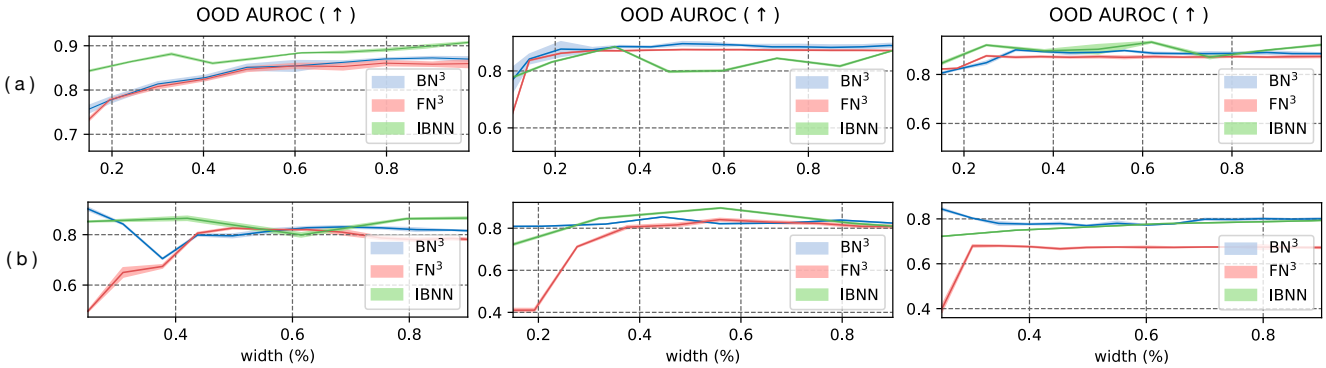

Figure 9: The AUROC of OOD on (a) Cifar10 (b) Tiny ImageNet datasets with VGG11, MobileNetV2 and ResNeXt-Cifar (left to right).

9.2. BN statistics

We show that collecting batch normalization statistics on a small training set present similar performance to using the whole dataset In this example, we use VGG11 on Cifar10. The collection proceeds by forward the network by 2 iterations, with a batch size 512. Thus, in total, $1024/50000$ training data are used for statistics collection. The results are shown in Figure 8 as BN^3*. We can observe that this results are similar to using all training data for statistics collection, with slightly larger variance using a lower width.

9.3. OOD detection

For out-of-domain detection, we use the SVHN dataset as the OOD data [5]. The OOD detection performance with AUROC metric is shown in Figure 9. The performance is similar to that of AUPR in Figure 5.

9.4. Cifar100 results

The results on Cifar100 is shown in Figure 10. As the hyper-parameters are mostly from training on Cifar10, the results may not be optimal. We do not show the comparisons for MobileNetV2 here, as it is observed that IBNN-MobileNetV2 fails provide a decent performance on Cifar100, similar to Figure 5(b). The proposed BN^3performs well steadily on every task.

[5] http://ufldl.stanford.edu/housenumbers/

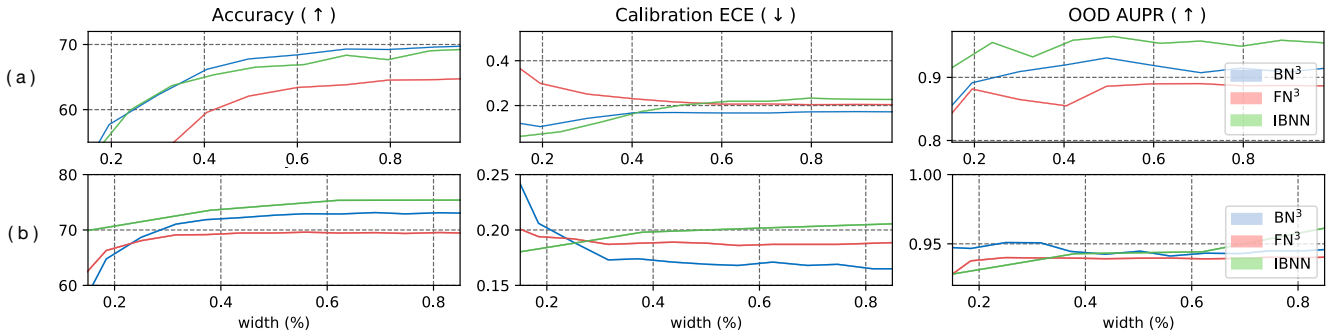

Figure 10: Results on Cifar100 for (a) VGG11, (b) ResNeXt-Cifar.