Towards Scalable Bayesian Deep Learning with Sampled Branching Networks

Abstract

We present Branch-SWAG, a method to perform approximate Bayesian inference with deep learning models in a scalable and efficient manner via sampled branching networks. Recently, Stochastic Weight Averaging-Gaussian (SWAG) has been shown to be a simple and effective way to generate samples for Bayesian model averaging of neural networks. We empirically demonstrate that by sampling only a subset of the neural network weights, one can generate branching networks that not only decrease the computational cost, but also improve the quality of the Bayesian estimates in terms of test loss and calibration. The experiments demonstrate that our proposed branching network method achieves state-of-the-art performances on image classification tasks.

1 Introduction

Quantifying how much a model knows or does not know is a critical aspect of machine learning. Particularly, when it comes to decision making based on machine learning predictions or direct decision output from machine learning models (such as policy or value functions in reinforcement learning), the understanding of confidence in models can have crucial effect on the performance or outcome of applications to which these machine learning methods are applied (e.g. medical diagnoses or autonomous driving).

Recent advances in deep learning methods have attracted many real-world applications. With powerful representations, deep learning methods are capable of learning to map from high-dimensional input data to an output. However, these mapping are often assumed to be overly confident or even taken blindly without consideration of confidence at all. Uncertainty quantification in deep learning is challenging, but fundamental not only to the research community but also among the practitioners because of its practical usage.

Bayesian deep learning has become an attractive area of research, not only due to its potential for improved predictive performance and generalization, but also because of its ability to provide predictions with uncertainty estimates instead of a single point estimate. One of the major drawbacks of Bayesian deep learning, however, is the computational cost involved in Bayesian model averaging due to the high-dimensional parameter space of state-of-the-art neural networks. This currently renders many Bayesian approaches for deep learning less scalable, thus impractical for many applications, especially when considering the recent and likely continuing trend of bigger models and larger datasets.

Our contributions are as follows:

- In this work, we propose *Branch Stochastic Weight Averaging-Gaussian* (Branch-SWAG) which is a scalable approximate Bayesian inference framework for deep learning. Our proposed framework is efficient both in terms of computation and memory requirements while achieving state-of-the-art performances.
- Branch-SWAG builds on SWAG [1] which computes the first and second momenta of stochastic gradient descent (SGD) iterates to estimate a Gaussian posterior of the neural network

weights. Instead of sampling an entire network as proposed in [1], we propose a sub-sampling technique that generates a stochastic branching network which is more efficient in terms of computation and memory requirements. This is crucial for scalability since many modern deep learning networks used in practice are extremely large in size [2, 3] and sampling multiple instances of these may not be even feasible.

- Our finding in numerical experiments is that the utilization of branching networks which sub-sample only a final portion of a large neural network leads to more accurate and better calibrated Bayesian model averages for various image classification tasks than sampling the entire neural network.

2 Background

2.1 Bayesian Neural Networks

We briefly discuss the Bayesian neural network methods, Let $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ be a collection of realizations of i.i.d random variables, where $\boldsymbol{x}_i$ is an image, $\boldsymbol{y}_i$ is a corresponding density map, and N denotes the sample size. In Bayesian neural network framework, rather than thinking of the weights of the network as fixed parameters to be optimized over, it treats them as random variables, and so we place a prior distribution $p(\theta)$ over the weights of the network $\theta \in \theta$. This results in the posterior distribution

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} = \frac{\left(\prod_{i=1}^N p(y_i|x_i,\theta)\right)p(\theta)}{p(\mathcal{D})}.$$

While this formalization is simple, the learning is often challenging because calculating the posterior $p(\theta|\mathcal{D})$ requires an integration with respect to the entire parameter space Θ for which a closed form often does not exist. [4] proposed a Laplace approximation of the posterior. [5] introduced the Hamiltonian Monte Carlo, a Markov Chain Monte Carlo (MCMC) sampling approach using Hamiltonian dynamics, to learn Bayesian neural networks. This yields a principled set of posterior samples without direct calculation of the posterior but it is computationally prohibitive. Another Bayesian method is variational inference [6, 7, 8, 9] which approximates the posterior distribution by a tractable variational distribution $q_\eta(\theta)$ indexed by a variational parameter η. The optimal variational distribution is the closest distribution to the posterior among the pre-determined family $Q = \{q_\eta(\theta)\}$. The closeness is often measured by the Kullback-Leibler (KL) divergence between $q_\eta(\theta)$ and $p(\theta|\mathcal{D})$. While these Bayesian neural networks are the state of art at estimating predictive uncertainty, these require significant modifications to the training procedure and are computationally expensive compared to standard (non-Bayesian) neural networks.

[10] proposed using Monte Carlo dropout to estimate predictive uncertainty by using dropout at test time. There has been work on approximate Bayesian interpretation of dropout [10, 11, 12]. Specifically, [10] showed that Monte Carlo dropout is equivalent to a variational approximation in a Bayesian neural network. With this justification, they proposed a method to estimate predictive uncertainty through variational distribution. Monte Carlo dropout is relatively simple to implement leading to its popularity in practice. Interestingly, dropout may also be interpreted as ensemble model combination [13] where the predictions are averaged over an ensemble of neural networks. The ensemble interpretation seems more plausible particularly in the scenario where the dropout rates are not tuned based on the training data, since any sensible approximation to the true Bayesian posterior distribution has to depend on the training data. This interpretation motivates the investigation of ensembles as an alternative solution for estimating predictive uncertainty.

2.2 Stochastic Weight Averaging-Gaussian (SWAG)

Our proposed method is based on the idea of Stochastic Weight Averaging-Gaussian (SWAG) [1]. SWAG is a simple, but effective approach for Bayesian deep learning with low additional cost during training compared to standard neural network training. It approximates a low-rank Gaussian posterior of the neural network weights using the first and second moment of SGD iterates. In [1], the authors demonstrate that SWAG achieves higher test performance compared to many state-of-the-art Bayesian deep learning approaches, including MC-Dropout [10], temperature scaling [14], SGLD [15], KFAC-Laplace [16] and SWA [17] on a range of neural network architectures and datasets for image classification.

2.3 Branching Network

Branching networks are neural networks that consist of a shared network or trunk and multiple structurally similar network branches that are connected to the trunk. Since a large portion of the network can be shared among all branches through the trunk, branching networks provide a more efficient way to learn and predict multiple outputs for the same given input than neural network ensembles. The trade-off is that the predictions of the branches are not fully independent like ensemble models. However, in applications such as computer vision, it is generally considered that early layers in an image recognition network tend to learn fundamental features, which we may not expect to vary much from model to model. With this in mind, the shared part of a branching network can be viewed as a feature extraction step followed by fully independent branches. Branching networks have been successfully applied in reinforcement learning or computer vision settings to efficiently train bootstrap ensembles that can approximate Bayesian inference in a non-parametric fashion [18, 19, 20].

3 Methodology

3.1 `Branch-SWAG` algorithm

We propose *Branch Stochastic Weight Averaging-Gaussian* (`Branch-SWAG`), in which we sample from subspaces based on the network architecture. Previous studies have found that low-dimensional subspace inference using `SWAG` on random subspaces, principal component spaces or mode-connecting subspaces can provide good estimates for Bayesian inference that are comparable to `SWAG` applied on the full model parameter space [21].

Algorithm 1: Training of `Branch-SWAG`

Input: $\mathcal{D}$: batched training data with inputs x and labels y; f_θ: pretrained neural network with parameters θ; η: learning rate; d: moment update interval; K: rank of covariance estimate

Choose branchout point and divide neural network f_θ into trunk τ_{θ_T} and branching part κ_{θ_B}, such that $f_\theta = \tau_{\theta_T} \circ \kappa_{\theta_B}$ and $\theta = (\theta_T, \theta_B)$

Initialize $\mathcal{Q} \leftarrow$ empty queue; $k \leftarrow 0$; $n \leftarrow 0$

for episode $i \leftarrow 1$ **to** N **do**
  foreach batch $(x, y) \in \mathcal{D}$ **do**
    $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(f_\theta(x), y)$
    if k mod $d = 0$ **then**
      $\bar{\theta} \leftarrow \frac{n\bar{\theta} + \theta}{n+1}$
      $\overline{\theta_B^2} \leftarrow \frac{n\overline{\theta_B^2} + \theta_B^2}{n+1}$
      $n \leftarrow n + 1$
      if size$(\mathcal{Q}) = K$ **then**
        $\mathcal{Q}$.dequeue()
      end
      $\mathcal{Q}$.enqueue(θ)
    end
    $k \leftarrow k + 1$
  end
end

Result: $\bar{\theta}$: first moment estimate of parameters (SWA estimate); $\overline{\theta_B^2}$: second moment estimate of parameters; $\mathcal{Q}$: queue containing last K snapshots of branch parameters θ_B

In our proposed framework, we adapt `SWAG` on a contiguous subset of layers that always include the final layer of the network. This choice is motivated by the idea of branching networks with the aim to reduce computational cost during inference. Sub-sampling from the end of a network can be viewed as generating a branching network that consists of a trunk, which is not sampled and uses `SWA` [17] estimate, and a subsequent set of branches sampled via `SWAG`. The branching architecture allows more efficient computation of samples, since the trunk is shared across all branches. It can also enable faster recalibration of batch normalization layers [22] in the sampled models, which is a

required step in SWAG to achieve good performance and can cause considerable slowdown in larger models and datasets [1].

The training procedure of Branch-SWAG is described in algorithm 1. It requires a pretrained network and a branchout point from which one chooses to branchout from. During the training phase we estimate the first moment of SGD iterates for all parameters and the second moment of SGD iterates only for the branching part of the network. In addition, we store K SGD iterates to estimate the covariance matrix of the Gaussian posterior as done in [1]. Once these estimates are obtained, we can use them to generate a branching network and perform inference as described in algorithm 2.

Algorithm 2: Inference of Branch-SWAG

Input: $\mathcal{X}$: batched input samples, $\mathcal{X}_{BN}$: batched training samples for batch normalization update, $\bar{\theta}$: first moment estimate of parameters (SWA estimate); $\overline{\theta_B^2}$: second moment estimate of parameters; $\mathcal{Q}$: queue containing last K snapshots of branch parameters θ_B; S: number of branch samples; neural network divided into trunk τ_{θ_T} and branching part κ_{θ_B}

Define $(\bar{\theta}_T, \bar{\theta}_B) \leftarrow \bar{\theta}$

Convert $\mathcal{Q}$ to matrix $\hat{D}$

$\hat{D} \leftarrow \hat{D} - \bar{\theta}_B$

Define $\Sigma_B \leftarrow \frac{1}{2}(\overline{\theta_B^2} - \bar{\theta}_B^2) + \frac{1}{2(K-1)}\hat{D}\hat{D}^T$

Set trunk parameters to $\bar{\theta}_T$

Update batch normalization statistics of trunk using $\mathcal{X}_{BN}$

Store trunk features of batch normalization step $\mathcal{X}'_{\mathcal{BN}} = \tau_{\bar{\theta}_T}(\mathcal{X}_{BN})$

Precompute trunk features of input samples $\mathcal{X}' = \tau_{\bar{\theta}_T}(\mathcal{X})$

for $j \leftarrow 1$ **to** S **do**

  Sample $\theta_B^{(j)} \sim \mathcal{N}(\bar{\theta}_B, \Sigma_B)$

  Set branch parameters to $\theta_B^{(j)}$

  Update batch normalization statistics of branch using $\mathcal{X}'_{\mathcal{BN}}$

  $p_j(y|\mathcal{X}) \leftarrow \kappa_{\theta_B^{(j)}}(\mathcal{X}')$

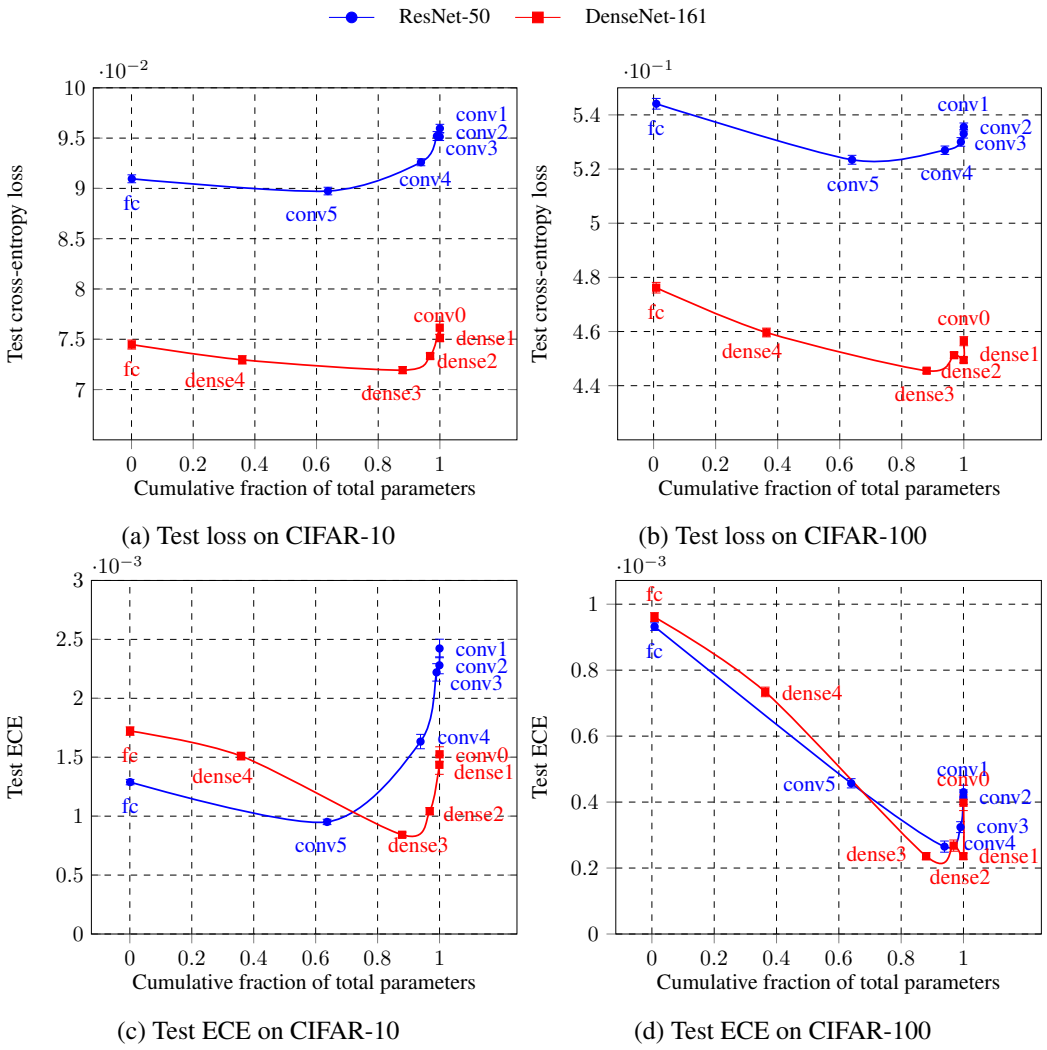
end

Result: Predicted probabilities $p_j(y|\mathcal{X}), j = 1, \ldots, S$

3.2 Uncertainty quantification and calibration

Calibration models are often required to improve the reliability of machine learning models, including neural networks, since the confidence or distribution of the prediction often does not match the empirically observed one. Based on our knowledge, calibration models for classification models so far have mostly been applied to align the predicted class probability (frequently referred to as confidence) with the observed accuracy [23, 14, 24]. However, this does not provide an uncertainty estimate of the predicted probability itself. In order to obtain an uncertainty estimate of the predicted probability itself, we fit a Beta distribution $\mathcal{B}(\alpha, \beta)$ to model the posterior probability for each class given multiple predictions of class probabilities. The Beta distribution parameters α and β are estimated using method-of-moments. It is to note that if the joint distribution for all classes follows a Dirichlet distribution, the marginal probability distribution for a given class c is Beta distributed, so the fitted posterior of a given class $p_c(Y = c|X)$ represents the binary probability whether a sample X belongs to class c versus all other classes.

To calibrate the fitted Beta distribution, we apply an isotonic regression model [25] trained on the validation dataset in each cross-validation fold. Since the Beta distribution can drastically vary in shape based on its parameters (positively or negatively skewed, unimodal or bimodal) we apply the following measures: 1) we mirror all negatively skewed distributions ($\alpha < \beta$) to be positively skewed ($\alpha > \beta$) and 2) we fit two separate calibration models, one for unimodal ($\alpha \geq 1$ or $\beta \geq 1$) and one for bimodal ($\alpha < 1$ and $\beta < 1$) distributions.



(a) Test loss on CIFAR-10

(b) Test loss on CIFAR-100

(c) Test ECE on CIFAR-10

(d) Test ECE on CIFAR-100

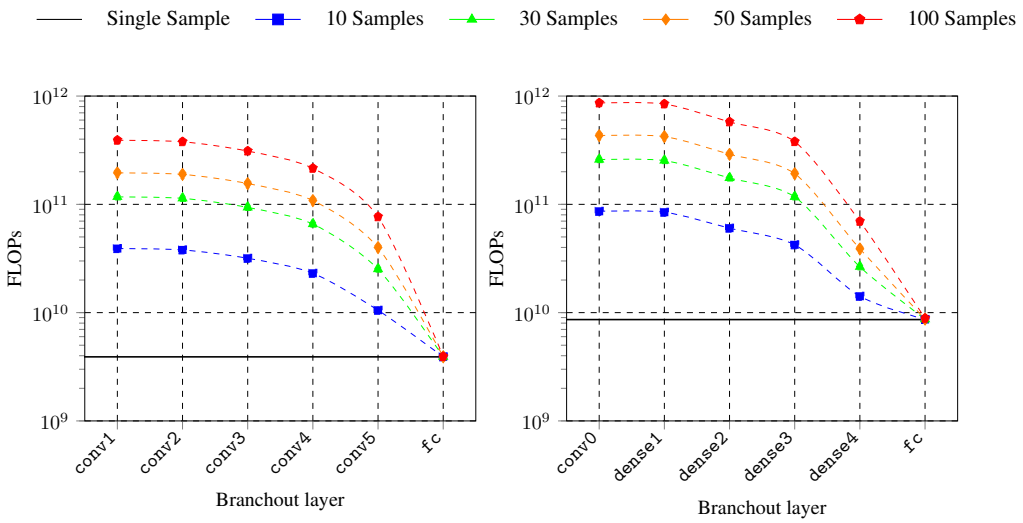
Figure 1: Test cross-entropy loss and expected calibration error (ECE) of branching networks generated via `Branch-SWAG` with different branchout layers. Cumulative fraction of total parameters refers to share of sampled parameters if viewed as continuously sampling all parameters from the end of the network until the branchout layer. Error bars represent standard error.

4 Experiments

4.1 Setup

We empirically evaluated our findings on the CIFAR-10 and CIFAR-100 dataset [26] using 5-fold cross-validation and 10 random seeds. We preprocessed all images during training with random horizontal flipping, resizing and brightness and contrast jittering. This data augmentation process was introduced to provide better generalization of the trained models.

In terms of model architecture, we investigated DenseNet-161 [27] and ResNet-50 [28] that were pretrained on ImageNet [29]. These two network architectures are comparable in number of parameters, but DenseNet-161 has higher computational complexity and yields better performance on benchmark image classification tasks. We fine-tuned all pretrained models for a total of 10 epochs, with the first 7 epochs using standard training and the last 3 epochs using `Branch-SWAG`. For standard training, we applied the Adam optimizer [30] with an initial learning rate of 10^{-4} that is halved every 5 epochs. For `Branch-SWAG` training, we used stochastic gradient descent (SGD) with a constant learning rate of 10^{-3} and stored the model parameters every 20 batches. Throughout all training



(a) ResNet-50

(b) DenseNet-161

Figure 2: Computational cost of single input inference of ResNet-50 and DenseNet-161 branching networks for different number of `Branch-SWAG` samples. The leftmost branchout layers in each plot (`conv1` and `conv0`) are equivalent to `SWAG` samples of the full network since they are the first layer in each network, respectively. In this figure, we assume a `fc` size of 1000.

we used a batch size of 32. We used a low rank approximation of rank 20 for estimating the `SWAG` covariance matrix and generated 30 branch samples to perform Bayesian model averaging similar to [1]. We also updated the batch normalization layers after sampling by passing the entire training data set through the network once as suggested in [17, 1].

To test the effect of branching out at different layers in the sampled network, we sub-sampled the network starting from different layers. We first identified for each network a set of potential branchout points, which we will refer to as branchout layers, based on their architecture. Table 1 and 2 contain the branchout layers considered for each network. For example, we used the following branchout layers for DenseNet-161: `conv0`, `denseblock1`, `denseblock2`, `denseblock3`, `denseblock4` and a fully-connected `fc` layer [27]. For each of the branchout layers, we then sampled all network parameters starting at the branchout layer until the final layer of the network. To give a specific example, sampling DenseNet-161 starting at `denseblock3` samples the parameters from `denseblock3` and all its subsequent layers (`denseblock4` and `classifier`). Thus, the resulting branching network after sampling will have branches starting at `denseblock3`.

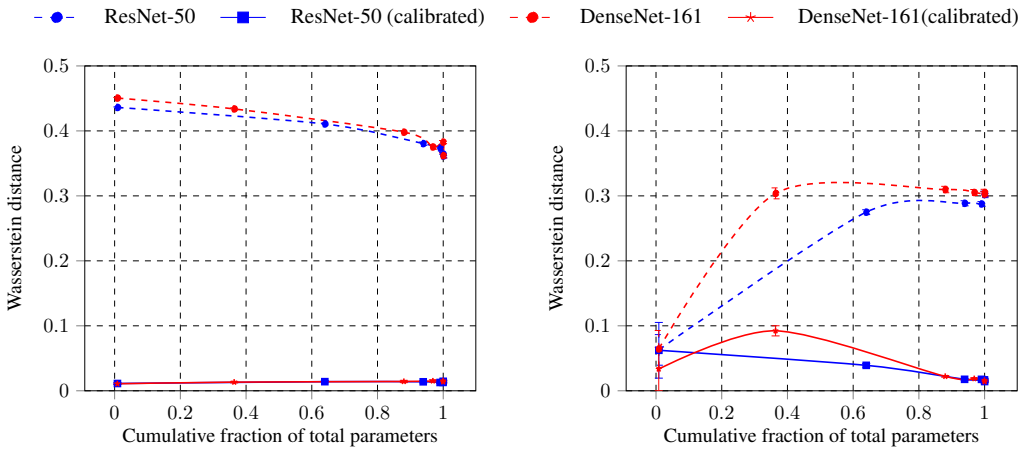
The experiments were run separately on multiple GPUs and took around one day per random seed and GPU, resulting in a total of 40 days of single GPU compute time.

4.2 Evaluation

We evaluated the models on the test dataset using the following performance metrics: 1) cross-entropy loss, 2) accuracy, 3) expected calibration error (ECE) [31, 14] and 4) the Wasserstein distance (WD) between the predicted and observed class probability distribution.

To estimate ECE, we bucketed the predicted class probabilities into 30 bins and computed the average of the absolute difference between the predicted probability and the observed accuracy within each bin weighted by each bins observation count. The bins were spaced logarithmically with smaller bin sizes towards the edges of the probability interval $[0, 1]$. This is to reduce the discretization error of binning because most predicted class probabilities will be close to 0 or 1 after sufficient training.

In addition to measuring the reliability of class probabilities via ECE, we also quantified the reliability of the fitted class probability distributions. We applied the same binning procedure as for ECE and estimated the observed probability distribution by computing the probability of the observed accuracy



(a) Unimodal distribution

(b) Bimodal distribution

Figure 3: Wasserstein distance of uncalibrated and calibrated branching networks with different branchout layers for CIFAR-100 test images. Wasserstein distance is measured between fitted, predictive cumulative distribution and observed cumulative distribution. Cumulative fraction of total parameters refers to share of sampled parameters if viewed as continuously sampling all parameters from the end of the network until the branchout layer. Error bars represent standard error.

based on the fitted Beta distribution for each class. We then computed the Wasserstein distance between the predicted and observed cumulative probability distribution.

4.3 Results

Sampling and branching out from different layers We found that the branching network generated by `Branch-SWAG` with sampling only a smaller fraction of the network, yielded the lowest test loss and ECE. Figure 1 shows the test loss and ECE as a function of sampling from different layers branchout and cumulative fraction of total parameters sampled. We use the term cumulative fraction to reflect the fact that we are continuously sampling parameters starting from the end of the network. We observe that the improvement of generalization measured by test loss is statistically significant as we sub-sample smaller parts of the network instead of the full network across both network architectures and datasets.

Uncertainty quantification and calibration While the test loss and ECE decreased as we sub-sampled a smaller part of the network, the calibration error of the fitted Beta distributions increased. However, the application of the isotonic regression calibration model drastically reduced the Wasserstein distance between the fitted cumulative distribution and the observed cumulative distribution, such that the differences resulting from sub-sampling different network parts became negligible. Figure 3 visualizes and compares the Wasserstein distance of the uncalibrated and calibrated models.

Upon further investigation we found that the fitted Beta distributions had a lower variance and kurtosis than the observed distributions. This could explain why applying `SWAG` on larger parts of the network decreased the Wasserstein distance since it can lead to a higher variance in the predictions and thus a wider Beta distribution.

Complexity analysis Figure 2 shows the computational inference complexity of branching networks generated by `Branch-SWAG` with different branchout layers. We can see that a ResNet-50 branching network branching out at `conv5`, which performed best in our experiment, can compute 30 sample predictions at a lower cost than 10 full `SWAG` samples, and that it can generate more than 100 samples at less cost than 30 full `SWAG` samples. Similar observations can be made in the case of DenseNet-161 branching networks, although the improvements for the best performing version (`dense3`) are smaller since the branches still cover 80% of the parameters. It is also worth noting that one could create branching networks that only branch out at the very last `fc` layer with comparable performance to full `SWAG`, but at a much lower cost. We observe that increasing the number of samples for such branching

Table 1: CIFAR-10 test performance of sampled branching networks.

Model	Branchout layer	Test loss	Accuracy	ECE
ResNet-50	`conv1`	0.0960±0.0005	0.9698±0.0002	0.0025±0.0001
	`conv2`	0.0952±0.0004	0.9699±0.0002	0.0023±0.0001
	`conv3`	0.0953±0.0005	0.9701±0.0002	0.0023±0.0001
	`conv4`	0.0926±0.0004	0.9702±0.0002	0.0017±0.0001
	`conv5`	**0.0898**±0.0004	**0.9704**±0.0002	**0.0010**±0.0001
	`fc`	0.0910±0.0004	0.9703±0.0002	0.0013±0.0001
DenseNet-161	`conv0`	0.0752±0.0004	0.9764±0.0002	0.0016±0.0001
	`dense1`	0.0762±0.0007	0.9761±0.0002	0.0015±0.0001
	`dense2`	0.0734±0.0003	0.9764±0.0002	0.0011±0.0001
	`dense3`	**0.0720**±0.0004	0.9765±0.0002	**0.0009**±0.0001
	`dense4`	0.0730±0.0005	**0.9769**±0.0002	0.0016±0.0001
	`fc`	0.0745±0.0005	**0.9769**±0.0002	0.0018±0.0001

Table 2: CIFAR-100 test performance of sampled branching networks.

Model	Branchout layer	Test loss	Accuracy	ECE
ResNet-50	`conv1`	0.5356±0.0015	0.8408±0.0005	0.0005±0.0001
	`conv2`	0.5331±0.0016	0.8411±0.0005	0.0005±0.0001
	`conv3`	0.5301±0.0016	0.8413±0.0005	0.0004±0.0001
	`conv4`	0.5270±0.0016	0.8414±0.0005	**0.0003**±0.0001
	`conv5`	**0.5235**±0.0017	**0.8428**±0.0005	0.0005±0.0001
	`fc`	0.5442±0.0020	0.8427±0.0005	0.0010±0.0001
DenseNet-161	`conv0`	0.4565±0.0017	0.8650±0.0004	0.0004±0.0001
	`dense1`	0.4495±0.0011	0.8654±0.0004	**0.0003**±0.0001
	`dense2`	0.4513±0.0012	0.8655±0.0004	**0.0003**±0.0001
	`dense3`	**0.4456**±0.0010	0.8659±0.0004	**0.0003**±0.0001
	`dense4`	0.4597±0.0017	**0.8661**±0.0004	0.0008±0.0001
	`fc`	0.4762±0.0020	**0.8661**±0.0004	0.0010±0.0001

network does not lead to a considerable increase in computational cost and that one could obtain a large number of samples at a comparable cost to a standard neural network. This demonstrates the potential gain in computational efficiency of branching networks generated by `Branch-SWAG`.

5 Discussion

We propose a method for scalable approximate Bayesian deep learning, which we refer to as Branch Stochastic Weight Averaging-Gaussian (`Branch-SWAG`). This method enables efficient sampling and computation of Bayesian model averages for large deep learning models while achieving state-of-the-art performances. We empirically demonstrated that our method can generate predictions for Bayesian model averaging at a substantially lower computational cost while achieving comparable state-of-the-art performances.

While we think the results look promising in pushing us one step closer towards scalable Bayesian deep learning, there are limitations of our work. One limitation is that we have only tested our method on image classification tasks. Our future work is to investigate whether our findings generalize to other machine learning domains such as natural language processing and reinforcement learning. Another limitation is that we have only considered branchout points based on the high-level neural network architecture and that they are not necessarily evenly distributed in terms of parameter coverage. We believe it is possible to obtain better results by considering more granular branchout points that in general can be at any elementary layer of the neural network. However, it will remain a challenge to find the optimal branchout point. While it seems that there is a range of branchout points in which the network performs better based on our experiments, it is difficult to find the exact optimum. The optimal branchout point varies by network architecture and is a hyperparameter to be tuned via a

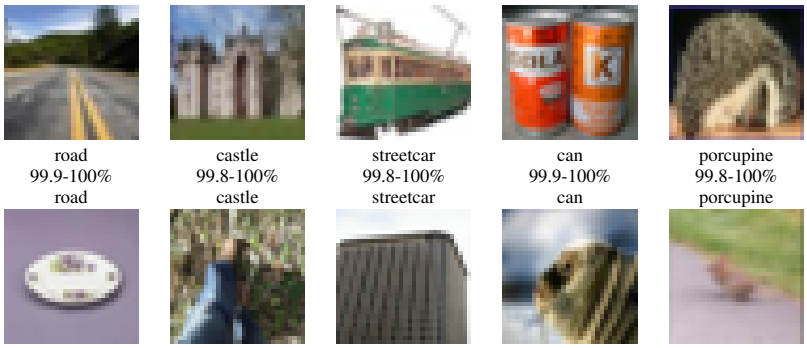


Figure 4: Uncertainty estimate examples of ResNet-50 with 30 sampled branches starting at `layer4`. Top: Five randomly sampled images from the top 20% CIFAR-100 test images with the highest 95% confidence interval (CI) lower bound. Bottom: Five randomly sampled images from the bottom 20% CIFAR-100 test images with the lowest 95% confidence interval (CI) lower bound.

meta-algorithm. Efficiently identifying the optimal branchout point in an adaptive fashion is one of the directions we consider for future research.

Overall, our findings suggest that the importance of model parameters for Bayesian inference in deep neural networks is structurally dependent. We generally find that network weights in early layers are less important than weights in final layers for Bayesian inference in computer vision network architectures. This would support the idea that early layers in an image recognition network are more responsible for feature extraction, while final layers are more responsible for classification. We might not expect feature extraction layers to vary much from model to model, whereas most of the model uncertainty could be about how the extracted features are used for classification in the final layers. We conjecture that this insight could guide future directions of scalable Bayesian deep learning research.

References

[1] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

[5] Radford M Neal. Bayesian learning via stochastic dynamics. In *Advances in neural information processing systems*, pages 475–482, 1993.

[6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

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

[8] Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, pages 1708–1716, 2016.

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

[10] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1050–1059. JMLR.org, 2016.

[11] Diederik 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.

[12] Shin-ichi Maeda. A bayesian encourages dropout. *arXiv preprint arXiv:1412.7003*, 2014.

[13] 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.

[14] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR, 2017.

[15] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 681–688, Madison, WI, USA, 2011. Omnipress.

[16] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations*. OpenReview.net, 2018.

[17] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *CoRR*, abs/1803.05407, 2018.

[18] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[19] Min-hwan Oh and Garud Iyengar. Sequential anomaly detection using inverse reinforcement learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1480–1490, 2019.

[20] Min-hwan Oh, Peder Olsen, and Karthikeyan Natesan Ramamurthy. Crowd counting with decomposed uncertainty. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):11799–11806, Apr. 2020.

[21] Pavel Izmailov, Wesley J. Maddox, Polina Kirichenko, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Subspace inference for bayesian deep learning. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 1169–1179. PMLR, 22–25 Jul 2020.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[23] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2801–2809. PMLR, 2018.

[24] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 625–632, New York, NY, USA, 2005. Association for Computing Machinery.

[25] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, page 694–699, New York, NY, USA, 2002. Association for Computing Machinery.

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[27] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[31] Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2901–2907. AAAI Press, 2015.

A Appendix

Table 3: Uncertainty reliability of sampled branching networks in CIFAR-10.

Model	Branchout layer	WD unimodal	WD unimodal (calibrated)	WD bimodal	WD bimodal (calibrated)
ResNet-50	`conv1`	**0.3707**±0.0021	**0.0334**±0.0032	0.2817±0.0044	**0.0341**±0.0023
	`conv2`	0.3732±0.0019	0.0363±0.0031	0.2924±0.0076	0.0361±0.0024
	`conv3`	0.3742±0.0018	0.0387±0.0034	0.2943±0.0057	0.0411±0.0033
	`conv4`	0.3806±0.0019	0.0413±0.0036	0.2788±0.0054	0.0421±0.0029
	`conv5`	0.4097±0.0013	0.0385±0.0035	0.3717±0.0134	0.2545±0.0305
	`fc`	0.4227±0.0015	0.0407±0.0039	**0.0123**±0.0087	(**)
DenseNet-161	`conv0`	0.3974±0.0045	0.0317±0.0036	0.3492±0.0068	**0.0268**±0.0020
	`dense1`	**0.3955**±0.0036	0.0393±0.0034	0.3380±0.0079	0.0301±0.0022
	`dense2`	0.4073±0.0034	0.0384±0.0036	0.3294±0.0062	0.0356±0.0026
	`dense3`	0.4141±0.0033	0.0387±0.0038	0.2944±0.0058	0.0572±0.0046
	`dense4`	0.4395±0.0027	**0.0308**±0.0037	**0.2460**±0.0269	0.1932±0.0353
	`fc`	0.4465±0.0026	0.0341±0.0036	(*)	(*)

(*) No bimodal samples available during validation. This can happen if the predictions of all branches are similar. It seems to occur in cases where we only sample the last fully connected layer.
(**) There are bimodal samples in the validation set, but not enough to fit a calibration model since we require at least 10 observations per probability bucket for estimation (see section 4.2).

Table 4: Uncertainty reliability of sampled branching networks in CIFAR-100.

Model	Branchout layer	WD unimodal	WD unimodal (calibrated)	WD bimodal	WD bimodal (calibrated)
ResNet-50	`conv1`	**0.3613**±0.0027	0.0146±0.0017	0.3027±0.0038	**0.0143**±0.0010
	`conv2`	0.3645±0.0024	0.0130±0.0016	0.3023±0.0050	0.0169±0.0016
	`conv3`	0.3735±0.0020	0.0128±0.0014	0.2876±0.0037	0.0177±0.0011
	`conv4`	0.3805±0.0026	0.0139±0.0017	0.2886±0.0040	0.0177±0.0012
	`conv5`	0.4107±0.0013	0.0141±0.0017	0.2750±0.0038	0.0392±0.0029
	`fc`	0.4362±0.0009	**0.0113**±0.0013	**0.0630**±0.0236	0.0624±0.0429
DenseNet-161	`conv0`	**0.3620**±0.0039	0.0138±0.0018	0.3052±0.0050	**0.0140**±0.0010
	`dense1`	0.3828±0.0024	0.0155±0.0018	0.3028±0.0041	0.0161±0.0011
	`dense2`	0.3753±0.0036	0.0152±0.0018	0.3052±0.0047	0.0188±0.0014
	`dense3`	0.3981±0.0022	0.0144±0.0018	0.3098±0.0050	0.0222±0.0014
	`dense4`	0.4339±0.0016	0.0134±0.0012	0.3039±0.0085	0.0923±0.0079
	`fc`	0.4506±0.001	**0.0109**±0.0012	**0.0644**±0.0284	0.0335±0.0335