Know Where To Drop Your Weights: Towards Faster Uncertainty Estimation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Estimating epistemic uncertainty of models used in low-latency applications and Out-Of-Distribution samples detection is a challenge due to the computationally demanding nature of uncertainty estimation techniques. Estimating model uncertainty using approximation techniques like Monte Carlo Dropout (MCD), DropConnect (MCDC) requires a large number of forward passes through the network, rendering them inapt for low-latency applications. We propose Select-DC which uses a subset of layers in a neural network to model epistemic uncertainty with MCDC. Through our experiments, we show a significant reduction in the GFLOPS required to model uncertainty, compared to Monte Carlo DropConnect, with marginal trade-off in performance. We perform a suite of experiments on CIFAR 10, CIFAR 100, and SVHN datasets with ResNet and VGG models. We further show how applying DropConnect to various layers in the network with different drop probabilities affects the networks performance and the entropy of the predictive distribution.

1 Introduction

Deep neural networks are increasingly playing an important role in every industry. Their prowess as function approximators that are trainable using gradient-based optimization methods and as feature extractors for huge amounts of data has made them extremely successful in Computer Vision, Natural Language Processing, Reinforcement Learning, etc. Many products and in-production systems, which use Deep Learning as a back-end, deal with sensitive data and mission-critical subsystems. It is imperative that we are able to guarantee the safety of such systems and make them robust to faults [1]. Identifying instances where the network is uncertain about its follow through and quantifying model confidence is important to guarantee a fail-safe mechanism [2][3]. Once we identify that a network is unsure about its predictions, the control can be transferred to a human-in-the-loop to take over.

Modern neural networks are poorly calibrated [4], i.e, the probability associated with the predicted class label is not a good representative of the true correctness likelihood. Bayesian Neural Networks [5], [6] (BNN) combine the strengths of neural networks and stochastic modelling. Networks are usually trained with maximum likelihood (MLE) or Maximum a Posteriori (MAP) estimation. Instead of the point estimates that we get from both MLE and MAP, we impose a full posterior distribution over the parameters of the network that takes weight uncertainty into account. Finding the posterior analytically, however, is computationally intractable. Therefore, we approximate the true posterior with a variational distribution $q(w|\theta)$ by minimizing the Kullback-Leibler(KL) Divergence between the variational distribution (such as a Gaussian) and the true distribution $p(w|D)$, where D is the dataset.

Uncertainty estimation in neural networks is a compute-intensive process, even using simple approximations like Monte Carlo Dropout [7] and Monte Carlo DropConnect(MCDC)[8]. In this

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

paper, we propose a modified MCDC, that has significant computational gains over a vanilla MCDC implementation.

In this work we explore how to reduce the computational requirements to make forward passes of a DropConnect enabled network, and observe some interesting results. Applying DropConnect to a network improves its accuracy compared to a baseline without DropConnect, and the magnitude of the improvement varies with the number of layers using DropConnect. We expected that the performance in terms of uncertainty quality and accuracy would be the best with a fully DropConnect network, but our results show that using less DropConnect layers performs best, with minimal differences in the quality of uncertainty. We make the following contributions:

- We identify that there is a significant computational speedup in applying MCDC to only a select number of layers with marginal loss in uncertainty quality. We call this method Select-DC.
- We find that the best model performance in terms of accuracy happens with partial usage of DropConnect across the network, not with a full DropConnect one, which we believe is unexpected.
- In contrast to our expectation, we find that networks with DropConnect applied to select number of layers do not observe significant changes in uncertainty estimation quality.
- We characterize the trade-off between the number of layers MCDC is applied to, and model performance.
- We find that DropConnect can be enabled at inference on select layers from the network's output with minimal loss in accuracy and uncertainty quality, enabling dynamic use of DropConnect without network retraining.

1.1 Related Work

Neural networks with their large number of parameters render the task of predicting the posterior distribution intractable. There has been extensive research on Bayesian neural networks [6] [9] and Monte-Carlo sampling for uncertainty estimation in deep learning. While exact Bayesian inference is intractable due to computational costs and challenging inference, several studies have been conducted on approximate methods using deterministic approaches. [7] developed a theoretical framework and demonstrated the mathematical equivalence of Dropout training in an arbitrary neutral network with approximate Bayesian inference in deep Gaussian processes [10]. The prediction ensemble is generated by keeping drop-out at test time. Similar approximations can be done using DropConnect[8]. They also introduce an adaptive approach to model the irreducible noise using held-out validation. This proposed a scalable alternative to mean-field variational inference methods, such as Radial BNNs [11] and Bayes by Back-prop[12]. While these class of methods that use Monte-Carlo(MC) sampling work well with estimating the multi-modal distribution, they cannot represent data uncertainty. While a solution that fine-tunes dropout rates has been proposed [13], [14] discusses examples where this method fails to generate correct predictions.

Deep ensembles [15] is another frequentist approach towards modeling uncertainty by training multiple models with different random initializations, where each model's parameters could be interpreted as a sample from the underlying weight distribution. While this outperforms Bayesian methods trained using variational inference, it is computationally intensive, and the computation cost, at both train and test time, scales linearly with the number of ensembles. An alternative to this method is [16] that learns confidence estimates on the out-of-distribution detection task, without requiring labels for supervised training. The model architecture and loss function formulation is similar in implementation to uncertainty estimation for regression tasks as in [17], and [18]. There have also been attempts at the above task of out-of-distribution detection using generative models [19], but are computationally expensive than classification models and do not perform predictive uncertainty estimation.

2 Uncertainty Estimation using DropConnect

In this section, we briefly review DropConnect and Dropout. We explain how applying DropConnect to all layers approximates a Bayesian NN, and is used to model epistemic uncertainty (the measure of

what the model does not know). We then show that we can gain significant computational speedup in estimating uncertainty by applying DropConnect to a select few layers in the model, which we call **Select-DC**

2.1 DropConnect and Dropout

Consider a neural network with N layers. For simplicity, we assume this is a network with simple feed-forward layers. However, it can be easily extended to modern networks like ResNet[20] and VGG [21]. We denote the network parameters by θ, and the weight kernel of the layer by W. The oth column in the weight matrix denotes the oth neuron in the layer. We ignore bias for convenience, however, biases are not masked in our implementation.

DropConnect [22] randomly drops out individual weights in the weight matrices at every training step. The dropping is actually done by masking weights, i.e, zeroing out their activations. This forces the network to not over-rely on a specific connectivity pattern and adapt to various connectivity patterns. Let σ be an activation function. The outputs of a layer will then be

$$Y = \sigma(X(W \odot M))$$

Where $\odot$ is the Hadamard product, X the activations of the previous layer or the input vector if its the first layer, M is the binary mask that randomly drops out weights whose elements are $\sim$ Bernoulli (p). DropConnect is a generalization of dropping out entire neurons in a network, as in Dropout [7]. Dropout follows the same procedure as above, but instead samples a mask that randomly masks out entire columns of the weight matrix. Since a neuron is completely removed, Dropout is better written as

$$Y = \sigma((X * W) \odot M)$$

The activations of the corresponding neuron are zeroed out. DropConnect at inference has been shown to approximate the Bayesian predictive posterior distribution [8], similar to what Dropout [7] does at inference. These methods are called MC-Dropout and MC-DropConnect.

2.2 Why DropConnect on Select Layers ?

Let us consider the total time complexity of running inference through a trained network, where we apply DropConnect to each convolutional or dense layer. If the network has N layers, and a forward pass through each layer takes approximately M units of time, the total cost of one forward pass is $N * M$. If we run the sample through the network K times, the total time to calculate the statistics of the predictive distribution is $(K * N * M)$.

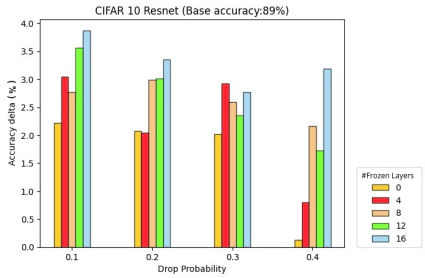
State-of-the-art neural networks are usually hundreds, of layers deep. Running multiple forward passes through these networks might be ill-suited for applications that demand low latency and have limited compute. If we apply DropConnect to a select L number of layers, the total time taken to compute K samples would be,

$$T = (N - L)M + (L * M * K) \leq (K * N * M) \tag{1}$$

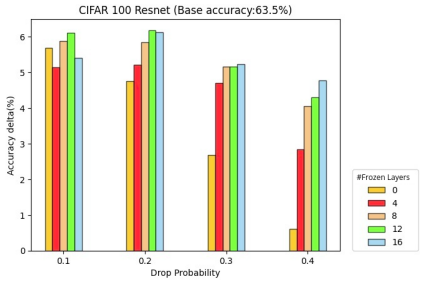
The advantage here is that we can vary the layers we apply DropConnect to during inference. We can also train our models using DropConnect applied to all layers(MCDC), and use Select-DC for inference. This flexibility implies that we are not limited to Select-DC if the uncertainty quality is not good enough. We can immediately shift to MCDC to gain the best performance.

Select-DC has one hyper-parameter λ, which is the number of layers without masked weights during inference. We term this subset of layers as the frozen block. The frozen block must begin from the input layer to satisfy the equation 1. For example, if $\lambda = 4$ in a ResNet20 model, DropConnect is not applied to layers 1-4, and the layers 1-4 form the frozen block. Instead, if we did not apply DropConnect to layers 1, 4, 8, 15 for instance, we cannot store the intermediate activations to reuse them to infer multiple samples.

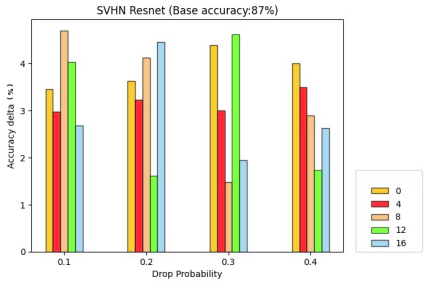
Dropping off weights from layers is an approximation of sampling from the underlying weight distribution. We expected that reducing the number of layers DropConnect is applied to, might reduce the uncertainty estimation quality compared to MCDC. Intuitively, if we apply DropConnect to all layers starting from layer x, then every layer before it is common to all forward passes used to estimate the predictive distribution- making these samples less diverse. However, to our surprise, we noticed that there is little to no loss in uncertainty estimation quality using Select-DC.



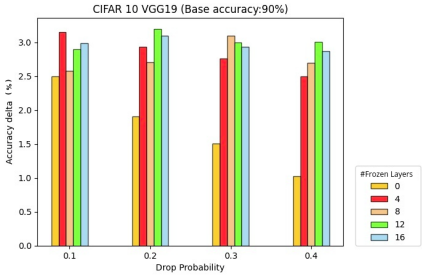
(a) ResNet 20 - CIFAR 10



(b) ResNet 20 -CIFAR 100



(c) ResNet 20 - SVHN



(d) VGG 19 - CIFAR 10

Figure 1: Comparison of ResNet20 and VGG19 performance on CIFAR10, CIFAR100, SVHN for varying λ and drop probabilities.

3 Experiments

We evaluate our proposed method on three datasets for image classification: CIFAR10, CIFAR100 and SVHN. For all the datasets, we use ResNet-20 [20] with random shifts and horizontal flips as data augmentation. We also use VGG19 [21] to further evaluate the performance of our method on CIFAR10. We report accuracy relative to a baseline model without uncertainty quantification, and all metrics are computed over 25 forward passes of each model.

3.1 Hardware and Setup

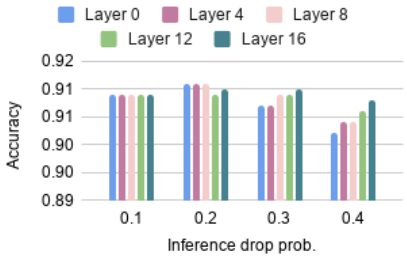
We ran all our experiments on a single machine with an NVIDIA P100 GPU. For both CIFAR 10 and CIFAR 100, all models were trained for 200 epochs with a batch size of 100, and SGD with Nesterov momentum as the optimizer. We varied the learning rate over the course of training, and we maintain a learning rate of 0.5 till 50% of train time. We then linearly decrease the learning rate from 0.5 to 0.0005 till 90% of train time. For the last 10%, we maintain the learning rate at 0.0005. We experimented with various learning rate schedules like linear, exponential decay and cyclical learning rates, but the described schedule performed best. All values reported (accuracy, NLL, entropy) are the mean of 25 samples (stochastic forward passes).

3.1.1 Select-DC on Training and Inference

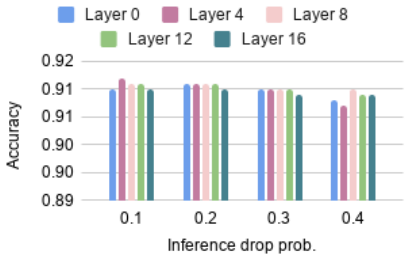
In this experiment, we train the models with $\lambda \geq 0$, and perform inference with the same setting. Fig. 1 illustrates the effect of changing λ on accuracy. A summary of our experiments on various datasets for combinations of different drop probability and λ shows that applying DropConnect to all layers is the least performing setting. This is as expected, since dropout is being applied to all weights in the network. As we decrease λ, we notice a steady improvement in accuracy.

3.1.2 MCDC on Training, Select-DC on Inference

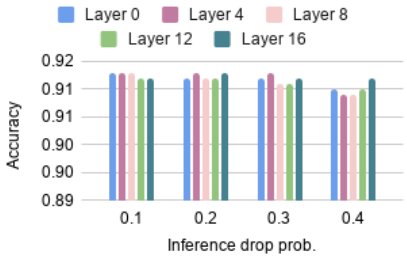
In this experiment, we train all of our models with $\lambda = 0$ but perform inference with varying values of λ. We observe no significant difference in the accuracy of the network on test data. Fig. 3 illustrates the uncertainty qualities of a model trained with MCDC, and Select-DC used at inference time across different drop probabilities, while Fig. 2 shows the corresponding accuracies. Fig. 3



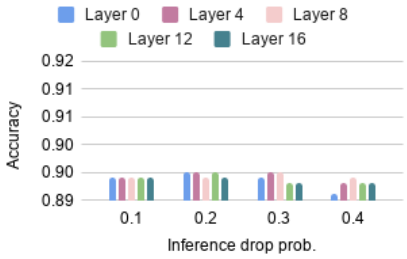
(a) Trained using MCDC with drop prob. 0.1



(b) Trained using MCDC with drop prob. 0.2



(c) Trained using MCDC with drop prob. 0.3



(d) Trained using MCDC with drop prob. 0.4

Figure 2: Accuracy Results of training the model using MCDC and inference using Select-DC for varying λ and drop probabilities on CIFAR-10.

shows that at lower drop probabilities, the entropy results are almost the same for all values of λ. As the drop probability at inference time increases, entropy decreases with increasing λ. These trends are consistent across varying drop probabilities applied during training. Fig. 4 further shows how the network's uncertainty changes as we apply a rotational transformation to images from the CIFAR10 dataset.

3.2 Computational Performance Analysis

Applying DC on less layers has a theoretical advantage over applying them on the full model. In this section we aim to evaluate this hypothesis and measure the speedup that can be obtained by using DC on select layers instead of the whole network. We estimate the number of floating point operations (FLOPS) as we vary λ. Fig. 5 shows how the total number of GFLOPS change with varying λ, and is consistent with the theory discussed in section 2.2. Increasing λ decreases the number of GFLOPS required for computation of a forward pass.

We plot the accuracy, negative log-likelihood and entropy as a function of GFLOPS for values of $\lambda \in (0, 21)$ to demonstrate the trade-off between error, uncertainty quality and computational requirements. In Fig. 6a, we observe that the accuracy decreases with an increase in GFLOPS, i.e, decreasing values of λ. This is consistent with the trends observed in Fig. 1. In Fig. 6c, we see that the uncertainty of the network, characterized by entropy of the predictive distribution, increases as we decrease λ, i.e, decreasing number of GFLOPS. However, the loss in quality of uncertainty modeling falls far slower than the required GFLOPS. Particularly for lower drop probabilities, this loss in quality is negligible. Therefore, according to the situational demands, one can convert Select-DC to MCDC to better model epistemic uncertainty.

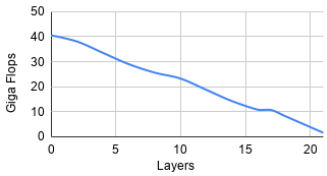
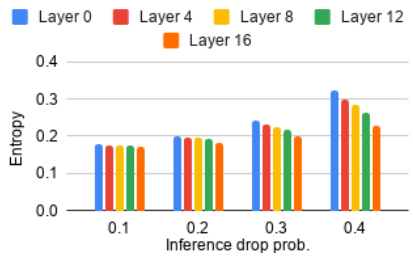
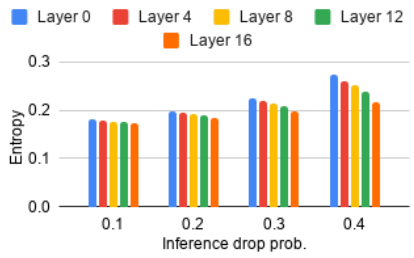


Figure 5: GFLOPS required for 25 forward passes for ResNet20 on CIFAR-10 versus variations of λ

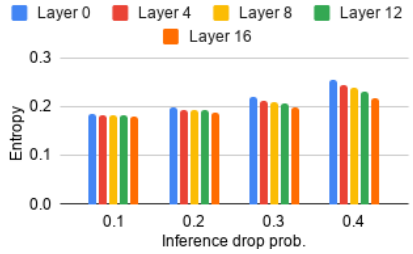
Our results from Fig. 6 show that while using Select-DC, increasing λ has the effect of decreasing the amount of compute (in GFLOPS). At the same time, it reduces the accuracy slightly, around $1 - 2\%$, depending on the drop probability. This is consistent with other results, where the network with less Bayesian capabilities works as an approximation of the full Bayesian network. It results in lower



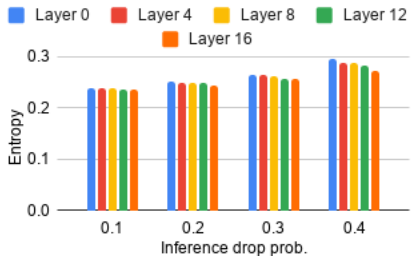
(a) Drop Connect on all layers with drop prob. 0.1



(b) Drop Connect on all layers with drop prob. 0.2

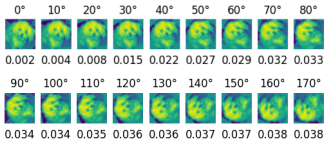


(c) Drop Connect on all layers with drop prob. 0.3

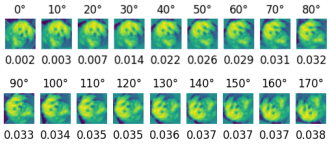


(d) Drop Connect on all layers with drop prob. 0.4

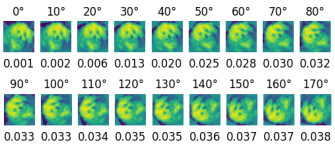
Figure 3: Results of training the model using MCDC and inference using Select-DC for varying λ and drop probabilities on CIFAR-10



(a) Drop prob 0.1, $\lambda = 0$



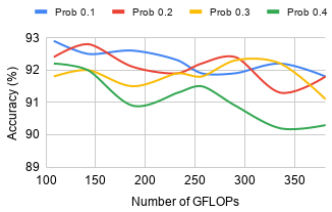
(b) Drop prob 0.1, $\lambda = 8$



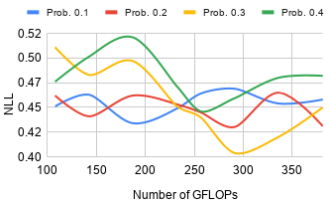
(c) Drop prob 0.1, $\lambda = 16$

Figure 4: Uncertainty estimation for rotated images in CIFAR10. We apply Dropconnect to all layers while training and use Select-DC for inference.

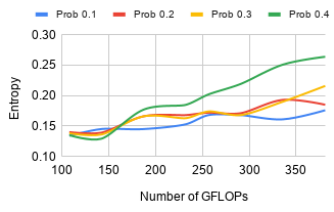
accuracy and NLL, and slightly increased entropy due to the loss in accuracy producing increased uncertainty.



(a) Accuracy vs Giga Flops



(b) NLL vs Giga Flops



(c) Entropy vs Giga Flops

Figure 6: Metrics measured against GFLOPS required to estimate the predictive distribution using 25 samples. The horizontal-axis in the figures correspond to the number of GFLOPS for increasing λ values. The GFLOPS decrease as λ increases.

Entropy of predictive distribution				
Drop probability	λ	ID Entropy	OOD Entropy	AUC
0.1	0	0.177	0.747	0.883
	4	0.161	0.765	0.890
	8	0.168	0.769	0.883
	12	0.145	0.759	0.893
	16	0.134	0.764	0.896
0.2	0	0.196	0.731	0.887
	4	0.193	0.738	0.877
	8	0.172	0.729	0.885
	12	0.166	0.723	0.878
	16	0.14	0.722	0.887
0.3	0	0.219	0.732	0.866
	4	0.19	0.730	0.868
	8	0.174	0.706	0.874
	12	0.166	0.732	0.871
	16	0.138	0.695	0.888

Table 1: Quantitative Out of Distribution results between CIFAR10 (ID) and SVHN (OOD).

3.3 Out-Of-Distribution Detection

We also show the Out-Of-Distribution (OOD) detection capabilities of our model. We train a model on the CIFAR10 dataset, with Select-DC, and evaluate on the SVHN test set for OOD samples. The image sizes are common in these datasets and they have no classes in common.

To classify a sample as in-distribution or OOD, we calculate the entropy of predictive distribution estimated through Monte Carlo sampling. The entropy is defined as,

$$H(x) = -\sum_{c \in C} f(x)_c log f(x)_c$$

We then classify all samples which result in entropy higher than a threshold as OOD, and in-distribution otherwise. An OOD input causes the network to output an approximately uniform distribution, which is observed from the high entropy values. Table 1 shows our quantitative results with different drop probabilities and λ values. Our results show that Select-DC can detect OOD samples nearly as well, if not exactly, as the models with full MCDC, with a difference of less than 1% AUC across different values of λ.

4 Conclusions and Future Work

In this work, we present the idea of applying DropConnect only to a select few layers instead of all layers in a neural network to model epistemic uncertainty. We show that we can achieve significant computational speedup by running the intermediate activations through the DropConnect applied part of the network without significant trade-off to uncertainty estimation quality. We show that this can also be used, without remarkable loss in performance, for Out-of-Distribution detection. We present and discuss how changing the subset of layers DropConnect is applied to affects the accuracy, NLL, entropy of a neural network. We experiment on CIFAR 10, CIFAR 100, and SVHN with ResNet and VGG models. We are excited to see how these observations extend to multiple domains like Natural Language Processing, or Reinforcement Learning.

Some limitations of SelectDC are the requirement that the frozen block must be at the beginning of the network, the unexpected loss of performance when the whole network uses DropConnect, which we believe requires further research, and we also expected larger differences in out of distribution detection performance, which might indicate that MC-DropConnect does not produce good epistemic uncertainty quantification.

References

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[2] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015.

[3] A. Loquercio, M. Segu, and D. Scaramuzza, "A general framework for uncertainty estimation in deep learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3153–3160, 2020.

[4] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *arXiv preprint arXiv:1706.04599*, 2017.

[5] J. S. Denker and Y. LeCun, "Transforming neural-net output levels to probability distributions," in *Advances in neural information processing systems*, 1991, pp. 853–859.

[6] D. J. MacKay, "A practical bayesian framework for backpropagation networks," *Neural computation*, vol. 4, no. 3, pp. 448–472, 1992.

[7] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.

[8] A. Mobiny, H. V. Nguyen, S. Moulik, N. Garg, and C. C. Wu, "Dropconnect is effective in modeling uncertainty of bayesian deep networks," *arXiv preprint arXiv:1906.04569*, 2019.

[9] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[10] A. Damianou and N. Lawrence, "Deep gaussian processes," in *Artificial Intelligence and Statistics*, 2013, pp. 207–215.

[11] S. Farquhar, M. A. Osborne, and Y. Gal, "Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning," *stat*, vol. 1050, p. 7, 2020.

[12] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *arXiv preprint arXiv:1505.05424*, 2015.

[13] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout," in *Advances in neural information processing systems*, 2017, pp. 3581–3590.

[14] I. Osband, "Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout," in *NIPS Workshop on Bayesian Deep Learning*, vol. 192, 2016.

[15] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in neural information processing systems*, 2017, pp. 6402–6413.

[16] T. DeVries and G. W. Taylor, "Learning confidence for out-of-distribution detection in neural networks," *arXiv preprint arXiv:1802.04865*, 2018.

[17] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" In *Advances in neural information processing systems*, 2017, pp. 5574–5584.

[18] P. Gurevich and H. Stuke, "Learning uncertainty in regression tasks by deep neural networks," *arXiv preprint arXiv:1707.07287*, vol. 17, 2017.

[19] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan, "Do deep generative models know what they don't know?" *arXiv preprint arXiv:1810.09136*, 2018.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[22] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International conference on machine learning*, 2013, pp. 1058–1066.

A Broader Impact

In security critical applications like autonomous driving, the perceptions models are usually trained on well-curated datasets, for example, with very good lighting and environment conditions. Even balancing the dataset by collecting samples of different conditions cannot cover all possible situations that the network might encounter. Here, uncertainty estimation and OOD detection is a pivotal requirement. However, a naive implementation of Bayesian NNs or even approximation techniques are computationally demanding. Our proposed method reduces computational requirements for uncertainty modeling and can be altered according to the requirements. For example, in the perception module of a self driving car, we can apply MCDC only to a few layers. During situations that are easy to interpret like broad day light, run the activations of the last frozen layer through the network to estimate samples of the predictive distribution. During night times, or times where the input to the perception module is noisy, we can apply MCDC to all layers in the network to get the best possible uncertainty estimates.

B DropConnect as Bayesian approximation

Similar to Mobiny et al. [8], in this section we prove that DropConnect approximates a Bayesian Neural Network. We provide the proof here for completeness. In a Bayesian NN with N layers, with weights $W = \{W\}_{i=1}^{N}$, our task is calculate the posterior distribution of the weights, $p = (W|D)$, given a dataset $D = (x, y)$. The predictive distribution of a label y' given a sample x' is,

$$\begin{aligned} p(y'|x', D) &= \mathbb{E}_{p(w|D)}[p(y'|x', w)] \\ &= \int p(y'|x', w)p(w|D)dw \end{aligned}$$

However, evaluating the integral for all weights in the weight space is clearly computationally intractable, and neither can it be evaluated analytically. Intuitively, this is equivalent to estimating the predictive distribution an infinite number of times, each time with a different weight configuration, and ensembling them to make a prediction. One way to approximate the posterior on the weights is to use variational inference. We use a variational distribution on the weights, $q_\theta(w)$ parameterized by θ, to minimize the Kullback-Leibler divergence between q and the true posterior. This is equivalent to minimizing negative evidence lower bound and takes the form,

$$L(\theta) = -\int q_\theta(w) log(p(y|x, w))dw + KL(q_\theta(w)||p(w)) \tag{2}$$

We can develop an accurate approximation, generalizing the approach followed in [7], using Monte Carlo sampling.

We approximate the variational distribution $q(w_k|\theta_k)$ for layer k as $w_k = \theta_k \odot M_k$, where M_k is the binary mask sampled from a Bernoulli distribution and θ_i, the variational parameters to be optimized.

Rewriting the first term in 2 as sum over all samples in the dataset,

$$-\int q_\theta(w) log(p(y|x, w))dw = \sum_{n=1}^{N} q_\theta(w) log(p(y|x, w)) = \frac{1}{N}\sum_{n=1}^{N} q_\theta(w) log(p(y|x, w')) \tag{3}$$

Applying DropConnect to weights can be interpreted as w', which is a sample from the weight distribution. The second term in 2 can be approximated using $\sum_{i=1}^{L} ||\theta||_2^2$ as shown in [7].

The loss function then finally takes the form,

$$\hat{L_{mc}} = \frac{1}{N}\sum_{n=1}^{N} q_\theta(w) log(p(y|x, w')) + \lambda \sum_{i=1}^{L} ||\theta||_2^2 \tag{4}$$

During inference, we can replace the posterior $p(w|D)$ with the approximate posterior $q_\theta(w)$ and use Monte Carlo sampling to approximate the integral.

$$p(y'|x', D) \approx \frac{1}{T}\sum_{t=1}^{T} p(y'|x', w'_t) \tag{5}$$

Each forward pass through the network generates a Monte Carlo sample from the posterior.