Complex & Intelligent Systems (2021) 7:2855–2869
https://doi.org/10.1007/s40747-021-00463-1

ORIGINAL ARTICLE

HatchEnsemble: an efficient and practical uncertainty quantification method for deep neural networks

Yufeng Xia[1] · Jun Zhang[2,3] · Tingsong Jiang[3] · Zhiqiang Gong[3] · Wen Yao[3] · Ling Feng[2]

Received: 23 April 2021 / Accepted: 3 July 2021 / Published online: 21 July 2021

Abstract

Quantifying predictive uncertainty in deep neural networks is a challenging and yet unsolved problem. Existing quantification approaches can be categorized into two lines. Bayesian methods provide a complete uncertainty quantification theory but are often not scalable to large-scale models. Along another line, non-Bayesian methods have good scalability and can quantify uncertainty with high quality. The most remarkable idea in this line is Deep Ensemble, but it is limited in practice due to its expensive computational cost. Thus, we propose *HatchEnsemble* to improve the efficiency and practicality of Deep Ensemble. The main idea is to use function-preserving transformations, ensuring HatchNets to inherit the knowledge learned by a single model called SeedNet. This process is called hatching, and HatchNet can be obtained by continuously widening the SeedNet. Based on our method, two different hatches are proposed, respectively, for ensembling the same and different architecture networks. To ensure the diversity of models, we also add random noises to parameters during hatching. Experiments on both clean and corrupted datasets show that *HatchEnsemble* can give a competitive prediction performance and better-calibrated uncertainty quantification in a shorter time compared with baselines.

Keywords Ensemble learning · Deep neural networks · Non-Bayesian method · Uncertainty quantification

Introduction

Deep neural networks (DNNs) have achieved the most advanced performance in various machine learning tasks [1] and are becoming more and more popular in the fields of computer vision [2], speech recognition [3], natural language processing [4], and bioinformatics [5]. Despite the excellent prediction performance, DNNs have difficulty in quantifying the uncertainty of their prediction. Recent studies have shown that DNNs are overconfident in their prediction results and produce miscalibrated softmax output probabilities for classification [6]. Moreover, they may make wrong and confident prediction for out-of-distribution samples that differ significantly from the training data distribution [7]. From self-driving cars to automatic medical diagnostics, uncertainty quantification has become an urgent need for many real-world applications, making it critical to equip DNNs with the ability to understand unknown information.

The existing neural network uncertainty quantification methods can be divided into two categories. The first category is based on Bayesian neural networks (BNNs) [8,9]. BNNs quantify predictive uncertainty by making the model parameters obey a probability distribution rather than using point estimates. Although BNNs provide a set of theoretical methods for uncertainty quantification, it is usually difficult to infer the true posteriors of the parameters. Moreover, specifying parameter priors for BNNs is challenging because the parameters of DNNs are huge in size.

✉ Wen Yao
wendy0782@126.com

Yufeng Xia
xiayufeng15@outlook.com

Jun Zhang
mcgrady150318@163.com

Tingsong Jiang
tingsong@pku.edu.cn

Zhiqiang Gong
gongzhiqiang13@nudt.edu.cn

Ling Feng
fengling@tsinghua.edu.cn

1 College of Aerospace Science and Engineering, National University of Defense Technology, Changsha 410073, China

2 Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

3 National Innovation Institute of Defense Technology, Chinese Academy of Military Science, Beijing 100000, China

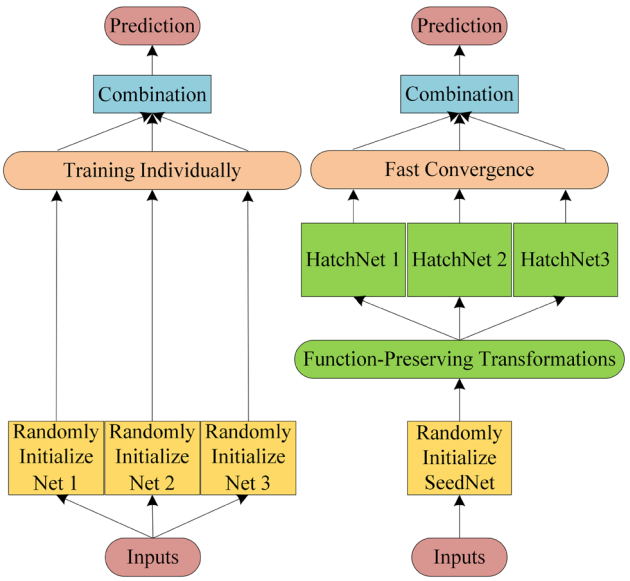


Fig. 1 Process comparison between standard Deep Ensemble (left) and HatchEnsemble (right)

Another category is based on non-Bayesian approaches and several methods have been proposed for uncertainty quantification. The most prominent idea in this category is model ensembling [10], which trains multiple DNNs with different initializations and uses all the prediction results for uncertainty estimation. Lakshminarayanan et al. [10] showed that Deep Ensemble gives reliable predictive uncertainty while remaining scalable and straightforward. Ovadia et al. [11] presented a large-scale evaluation of different methods for quantifying predictive uncertainty under dataset shift across different data modalities and architectures. They found Deep Ensemble seems to perform the best across most of the metrics and be more robust to dataset shift than other methods such as MC-Dropout [12] and temperature scaling [13].

However, the standard Deep Ensemble is limited in practice due to its computational costs, which increase linearly with the ensemble size. Each ensemble member needs an independent training process, which is time-consuming when the size of the model or dataset is large. A single neural network may take several days to train on some high-performance hardware, where the time cost is unacceptable [14–16]. In this paper, we propose *HatchEnsemble*, which solves the above problem and quantifies uncertainty with high quality. The process of the standard Deep Ensemble method and our method are shown in Fig. 1.

Summary of contributions. Our contribution in this paper is threefold.

- We propose an efficient and practical ensemble method called HatchEnsemble for ensembling neural networks. By reusing parameters in the smaller-sized SeedNet and transferring the knowledge it learned to HatchNets, the convergence speed of the HatchNets can be accelerated. Our method improves efficiency while retaining comparable result quality.
- Based on our method, we propose two kinds of hatches, which not only allow us to ensemble networks of the same architecture, but also allow us to ensemble networks with different widths.
- We propose a series of tasks for evaluating the quality of the predictive uncertainty, in terms of calibration in supervised learning problems. We show that our method (i) significantly outperforms MC-Dropout and (ii) matches Deep Ensemble but more faster.

Related work

In this section, we describe some related work about uncertainty quantification methods for deep neural networks, mainly divided into Bayesian neural networks and non-Bayesian neural networks.

Uncertainty quantification method based on Bayesian neural networks

In recent years, there have been many related works devoted to making deep neural networks contain probability characteristics to predict uncertainty. A large part of these works are based on Bayesian theory [17]. First, assume that the neural network parameters obey a certain prior distribution and then train the neural network through training data to calculate the posterior distribution on the parameters. Using this posterior distribution to quantify the uncertainty of the prediction. It is almost impossible to accurately infer the true posterior distribution through the Bayesian formula for models with large parameters such as neural networks. So a series of approximate inference methods are produced, including Laplace approximation [18], Markov Chain Monte Carlo (MCMC) [19], as well as recent works on variational Bayesian methods [20,21] and expectation propagation [22].

A key element that can affect the performance of BNNs is the choice of the prior distribution. The most common prior distribution to use is the independent Gaussian distribution, which can only give limited and even biased information for uncertainty. And because the Bayesian method involves sampling from the distribution, BNNs are more difficult to train, and the calculation is relatively slow. The experiment results in Ref. [11] also prove that BNNs are difficult to get to work on larger datasets such as ImageNet and other architectures such as LSTMs. Therefore, we are more inclined to study uncertainty quantification methods based on non-Bayesian theory.

Uncertainty quantification method based on non-Bayesian neural networks

Several non-Bayesian methods have also been proposed for uncertainty quantification. Gal et al. [12] proposed a simple non-Bayesian uncertainty quantification method called Monte-Carlo Dropout (MC-Dropout). By enabling dropout [23] in training and testing phases and making multiple forward passes through the network using the same input, one can easily estimate predictive uncertainty. Many works have used this method in recent years for its practicality. Some works [12,24,25] also tried to explain this method from the perspective of Bayesian theory.

Another non-Bayesian uncertainty quantification method is Deep Ensemble [10] mentioned in the previous section. More recently, Ovadia et al. [11] benchmarked existing methods for uncertainty modeling on a broad range of datasets and architectures and observed that ensembles tend to outperform variational Bayesian neural networks in terms of both accuracy and uncertainty. Gustafsson et al. [26] applied their proposed framework and provided the first properly extensive and conclusive comparison of ensembling and MC-Dropout, the results of which demonstrated that ensembling consistently provides more reliable and practically useful uncertainty estimation.

Recently, many works have been devoted to improving Deep Ensemble. Wen et al. [27] proposed *BatchEnsemble* by defining each weight matrix to be the Hadamard product of a shared weight among all ensemble members and a rank-one matrix per member. Lee et al. [28] proposed *TreeNets* and Asif et al. [29] used knowledge distillation to reduce model parameters. Snapshot Ensembles [30] use cyclic learning rate strategy to save models that converge to multiple local minima within a training period and then use them for ensembling. But most of these studies cannot support ensembling neural networks of different architectures, and only a small part considers the impact of the diversity of models. The focus of most existing methods is to improve the prediction performance of the model, and only a few of them are studying ensembling methods from the perspective of uncertainty quantification.

Method

Aiming at the problem of the high computational consumption of the Deep Ensemble, this part will introduce how our method solves this problem in detail. We first define two parts of our method: SeedNet and HatchNet in "Definition: SeedNet and HatchNet" section. Then in "Training procedure of HatchEnsemble" section, the Hatch method and the entire training process of HatchEnsemble are introduced. Two different hatch methods derived from our method will be described in "Two different Hatch methods" section. Finally, we describe how to increase the diversity between models in "Improving diversity via adding noises to parameters" section.

Definition: SeedNet and HatchNet

As shown in step 1 and step 3 of Fig. 2, SeedNet is a single network and can be seen as the foundation of HatchNets. HatchNets can be seen as the growth of a SeedNet. For a fully connected neural network, hatching means increasing the number of neurons in the same layer. For a convolutional neural network, hatching means increasing the number of channels in one layer.

Suppose a training dataset $\mathcal{D}$ consists of N i.i.d. samples $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$. A SeedNet is represented by a function $y = f(x;\theta)$ where θ is the parameters of the network, x is the input to the network, and y is the output of the network. Our hatch operation is to choose a new set of parameters θ_i' for HatchNets $y = h_i(x;\theta_i')$ such that

$$\forall x \in \mathcal{D},\ f(x;\theta) = h_i(x;\theta_i') \quad (i = 1, 2, ..., M) \tag{1}$$

where M represents the number of HatchNets; in other words, it also represents the number of ensemble members.

We call this process hatch, which means that different HatchNets are extended from the SeedNet.

Training procedure of HatchEnsemble

Training Step 1: Training the SeedNet. As shown in step 1 of Fig. 2, first, choose a standard basic neural network architecture as the SeedNet. On the one hand, the standard basic neural network structure is reusable, which is convenient for reusing and modification, increasing the practicability of the method proposed in this paper. On the other hand, using the standard basic neural network architecture as the SeedNet can facilitate comparison with the baseline proposed by other researchers. Then, train it with the entire data set until convergence. This allows the SeedNet to learn a good core representation of the data.

Training Step 2: Hatching ensemble networks. Once the selected SeedNet is trained well, the next step is to use a series of function-preserving transformations to generate HatchNets which are wider than the SeedNet. Function-preserving transformations mean to make some minor transformations based on preserving the neural networks function mapping relationship. It can ensure the knowledge learned by SeedNet is retained, and widening operation can ensure the diversity between HatchNets.

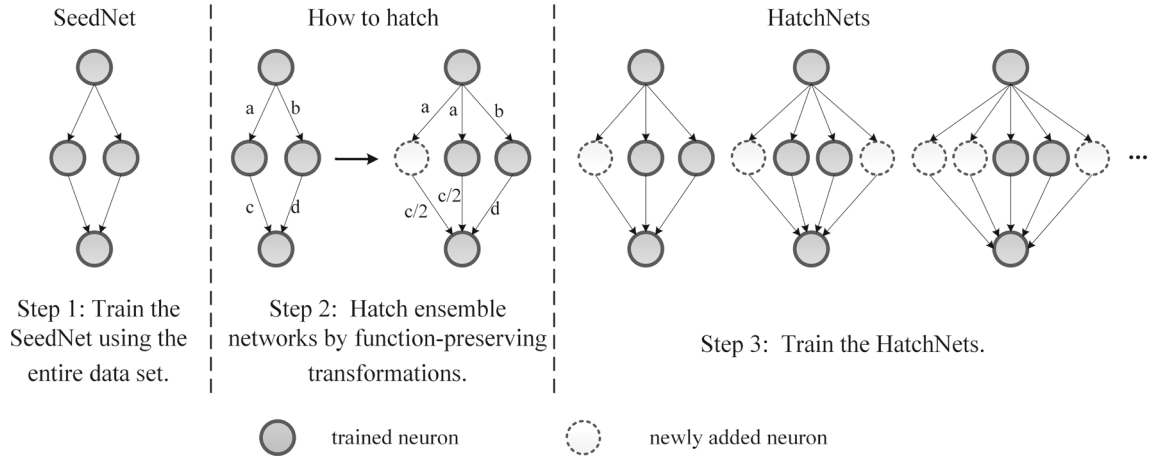


Fig. 2 HatchEnsemble trains an ensemble of neural networks by first training the SeedNet and transferring the function to the HatchNets. The ensemble networks are then further trained converging significantly faster than training individually

There are two methods to achieve Eq. 1: Network Morphism [31] derives sufficient and necessary conditions. When these conditions are met, the network will expand while maintaining its functions and provides an algorithm to solve these conditions. Net2Net, the other method, increases the capacity of a given network by adding an identification layer or keeping existing weights [32].

In HatchEnsemble, we adopt the second approach. Suppose that both layer i and layer $i+1$ are fully connected layers. To widen layer i, we need to replace the input-side weight matrix $\boldsymbol{W}^{(i)}$ for layer i and the output-side weight matrix $\boldsymbol{W}^{(i+1)}$ for layer $i+1$. If layer i has m inputs and n outputs, and layer $i+1$ has n inputs and p outputs, then $\boldsymbol{W}^{(i)} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{W}^{(i+1)} \in \mathbb{R}^{n\times p}$. Hatching allows us to replace layer i that originally had only n outputs with a layer that has q outputs, with $q > n$.

First, we need a random mapping function g, which randomly expands the list of ordinal numbers of n neurons $\{1, 2, \cdots, n\}$ to q neurons $\{1, 2, \cdots, q\}$, that satisfies

$$g(j) = \begin{cases} j & j \le n \\ \text{random sample from } \{1, 2, \cdots n\} & j > n \end{cases} \tag{2}$$

Through the mapping function $g(j)$, the first n items of the newly generated list are directly copied from the original list, and the nth to the qth items of the newly generated list are randomly selected from the original list.

Then, based on the random mapping function g, the new weight matrices $\boldsymbol{U}^{(i)}$ and $\boldsymbol{U}^{(i+1)}$ of these layers after the implementation of the hatch operation are given in the following form:

$$\boldsymbol{U}^{(i)}_{k,j} = \boldsymbol{W}^{(i)}_{k,g(j)}, \boldsymbol{U}^{(i+1)}_{j,h} = \frac{1}{|\{x \mid g(x) = g(j)\}|} \boldsymbol{W}^{(i+1)}_{g(j),h} \tag{3}$$

Algorithm 1: Pseudocode of the training procedure for our method

Input:
The number of HatchNets M, the architecture of the SeedNet
Output:
M different HatchNets
Initialize the SeedNet randomly
Use all dataset and appropriate loss function to train the SeedNet to convergence and then save it
for $m = 1 : M$ **do**
  Reuse the SeedNet and perform hatch operation on it to get the HatchNet m and its initial parameters θ_m
  Use all dataset and appropriate loss function to train the HatchNet m to convergence and then save it
end
Return M different HatchNets.

Here, the first n columns of $\boldsymbol{W}^{(i)}$ are copied directly into $\boldsymbol{U}^{(i)}$. Columns $n+1$ to q of $\boldsymbol{U}^{(i)}$ are created through a random strategy as defined in g. Each column of $\boldsymbol{W}^{(i)}$ is copied potentially many times. Because of this randomness, even neural networks with the same architecture and hatching can make their initial parameters different. For weights in $\boldsymbol{U}^{(i+1)}$, we must account for the replication by dividing the weight by replication factor given by $\frac{1}{|\{x|g(x)=g(j)\}|}$, similar to the operation on the output neurons in Dropout [23], to ensure that the output expectation is consistent with the original network.

Hatching is a process with low computational cost [31], which is negligible compared to training or testing neural networks, which also dramatically reduces the time-consuming of the entire pipeline.

Training Step 3: Training the HatchNets. Compared with initializing from scratch and training to convergence, the speed of further training of the HatchNets to convergence is significantly improved. The reason is that these ensemble networks' initialization parameters are derived from SeedNet

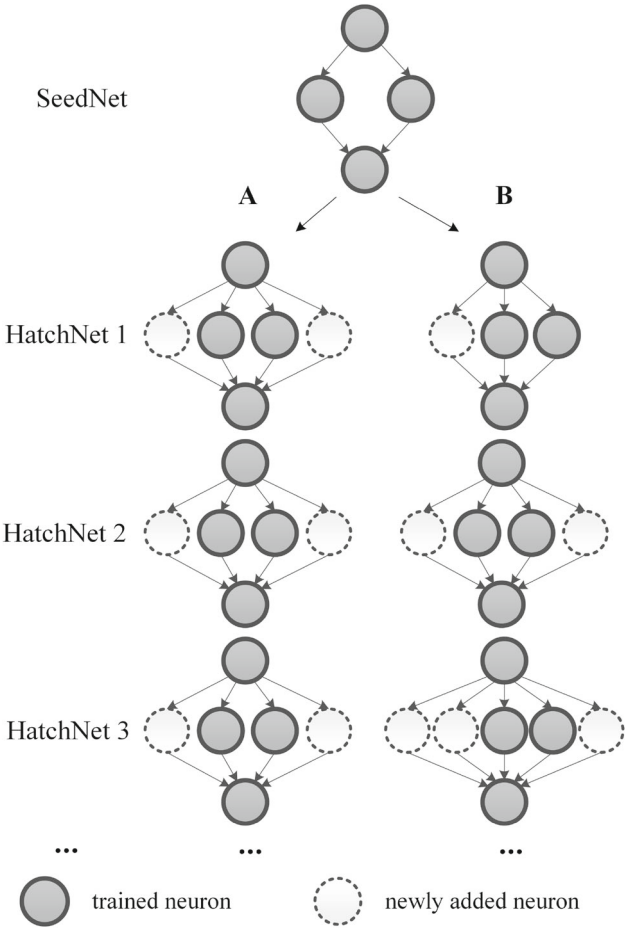


Fig. 3 Two different hatch processes. If the network structures of HatchNets are the same, we call this process *HatchEns A*; and if the network structures of HatchNets are different, we call this process *HatchEns B*

rather than random initialization parameters. The SeedNet has already converged in its own parameter space, so the ensemble networks only need to continue to explore a small part of the parameter space. Experiments have also confirmed that HatchNets can converge to local minimums with fewer epochs.

So far, the necessary training process is over. We can summarize it as Algorithm 1.

Two different Hatch methods

The ensemble method we propose can be divided into HatchEnsemble A and HatchEnsemble B according to whether the ensemble model's architectures are the same. We show it in Fig. 3. "A" represents that the model architectures transformed from the SeedNet are the same; that is, the way of widening is the same. Since the newly added parameters are randomly copied from the existing parameters, this operation can still ensure the models' diversity. "B" represents

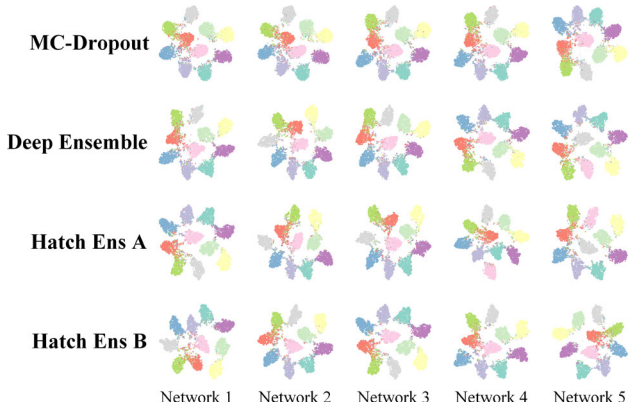


Fig. 4 t-SNE visualization results on the final hidden features of each individual network. The input is the test set of CIFAR-10

that the model architectures transformed from the SeedNet are different; that is, the way of widening is different. Since the number and value of the new parameters are different, this operation can also ensure the diversity of models.

Improving diversity via adding noises to parameters

In our method, the newly added neuron parameters or the newly added channel are randomly copied from the existing parameters. Due to the randomness of replication, the initial parameters of different HatchNets are different, increasing the diversity among them. To further amplify this advantage, we add Gaussian noise to the copied parameters so that the diversity between the HatchNets is amplified. This breaks symmetry after hatching, and it is a standard technique to create diversity when training ensemble networks. Further, adding noise forces the HatchNets to be in a different part of the hypothesis space from their SeedNet.

After training by the HatchEnsemble method, each network's prediction will tend to be more diverse and further leads to different feature distributions and decision domains as shown in Fig. 4. Figure 4 shows the t-SNE visualization results on each network's final hidden features in four methods. Different colors represent different categories. We can see that the final prediction results of multiple neural networks obtained by MC-Dropout method are very similar. The color distribution is roughly the same, which means the diversity is low. The color distribution of our proposed method is more random, which represents the diversity is better.

Experiments

In this section, we show the superiority of our proposed method by several experiments. We use these experiments to answer the following questions:

Q1. How accurate are the predictions, and how reliable are the uncertainty estimates of HatchEnsemble under **clean** datasets compared to other baselines?
Q2. How accurate are the predictions, and how reliable are the uncertainty estimates of HatchEnsemble under **corruptional** datasets compared to other baselines?
Q3. How efficient of HatchEnsemble compared with Deep Ens?
Q4. How diverse of neural networks in the HatchEnsemble?
Q5. How does the number of ensembles affect the performance of HatchEnsemble?

Preparation

Dataset

MNIST and **FashionMNIST** datasets consist of 70000 28×28 grayscale images in 10 classes, with 6000 images per class. There are 60000 training images and 10000 test images.

CIFAR10 dataset consists of 60000 32×32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

TinyImageNet dataset consists of 120000 64×64 color images in 200 classes, with 600 images per class. There are 100000 training images, 10000 test images and 10000 validation images.

CIFAR10-C and **TinyImageNet-C** datasets consist of 19 diverse corruption types applied to validation images of CIFAR10 and TinyImageNet. The corruptions are drawn from four main categories—noise, blur, weather, and digital. Each corruption type has five levels of severity since corruption can manifest itself at varying intensities. Figure 5 gives an example of the five different severity levels for shot noise. We test networks with CIFAR10-C and TinyImageNet-C images in our experiments, but networks should not be trained on CIFAR10-C and TinyImageNet-C. Networks should be trained on datasets such as CIFAR10 and TinyImageNet. Overall, the CIFAR10-C and TinyImageNet-C datasets consist of 95 corruptions, and all are applied to CIFAR10 and TinyImageNet validation images for testing a pre-existing network.

Experiment setting

In this part, we will explain our experimental setup in detail. Our experiments are mainly divided into five tasks:

- **Task1** evaluates LeNet [33] on MNIST. Model parameters were trained for 20 epochs. The basic LeNet architecture applies 2 convolutional layers (5×5 kernels of 6 and 16 filters respectively) followed by three fully-connected layers with two hidden layer of 128 and 64 activations.
- **Task2** evaluates LeNet [33] on FashionMNIST. Model parameters were trained for 40 epochs. The basic lenet architecture is the same as Task1.
- **Task3** evaluates VGG-11 [34] on CIFAR10. Model parameters were trained for 200 epochs. Training inputs were randomly distorted using horizontal flips and random crops preceded by 4-pixel padding as described in [14].
- **Task4** evaluates ResNet-18 [14] on CIFAR10. Model parameters were trained for 200 epochs. Data augmentation operation is the same as Task3.
- **Task5** evaluates ResNet-18 [14] on TinyImageNet. Model parameters were trained for 200 epochs.

For stochastic methods like MC-Dropout, we averaged 256 sample predictions to yield a predictive distribution, and dropout was applied before the final layer with $p = 0.1/0.2/0.5$. The size of the ensemble model (including the traditional Deep Ens and the ensemble method we proposed) was 5.

For all tasks, we use stochastic gradient descent with a mini-batch size of 128 for MNIST, FashionMNIST and CIFAR10 and a mini-batch size of 200 for TinyImageNet. All weights are initialized by sampling from a standard normal distribution. Training data are randomly shuffled before every training epoch. The initial learning rate is set to 0.01 for MNIST and FashionMNIST, 0.1 for CIFAR10 and TinyImageNet, respectively, and is divided by 10 at 45%, 67.5% and 90% of the total number of training epochs. To train the hatched neural networks, we change the above learning rate to half of the original to fine-tune the newly added parameters.

In consideration of extracting more features of the input image and enhancing the model's representation ability, our method's widening operation is applied to several layers close to the input in all models. Hatch Ens A ensembles the same model architectures and Hatch Ens B ensembles the different model architectures.

All experiments were run on the same server, using 4 NVIDIA TITAN RTX GPUs.

Metrics

In addition to metrics that do not rely on predicted uncertainty, such as classification accuracy ↑ (The arrow behind the metric represents which direction is better.), we propose three metrics to measure the quality of predicted uncertainty.

Negative Log Likelihood(NLL) ↓ is a standard measure of a probabilistic model's quality [35] and commonly used to evaluate the quality of model uncertainty. In deep learning, it is also called cross-entropy loss function [1]. Given a probabilistic model $\pi(Y \mid X)$ and n samples, NLL is defined as:

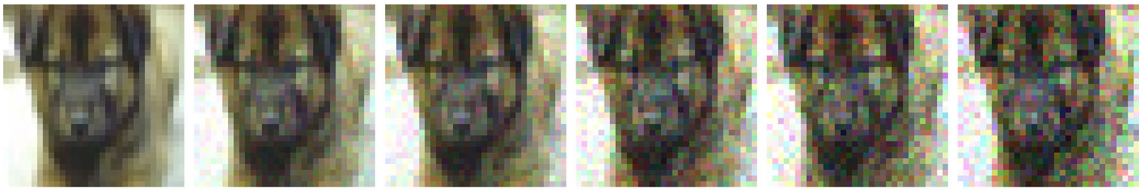


Fig. 5 Examples of CIFAR10 images corrupted by shot noise, at severities of 0 (uncorrupted image) through 5 (maximum corruption included in CIFAR10-C)

$$\mathrm{NLL} = -\sum_{i=1}^{n} \log\left(\pi\left(y_i \mid x_i\right)\right) \tag{4}$$

Brier Score(BS) ↓ is a proper score function that measures the accuracy of probabilistic predictions [36]. The drawback of the Brier score is insensitive to predicted probabilities associated with in/frequent events. It is obtained by calculating the mean square error of the true label y_i and the predicted probability p_i. The smaller the Brier score, the better the calibration effect. That is,

$$\mathrm{BS} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - p_i\right)^2 \tag{5}$$

Expected Calibration Error(ECE) ↓ measures the correspondence between predicted probabilities and empirical accuracy [37]. To calculate ECE, we group model predictions into S interval bins based on the predictive confidence (each bin has size $\frac{1}{S}$). Let B_s denote the set of samples whose predictive probability falls into the interval $\left(\frac{s-1}{S}, \frac{s}{S}\right]$ for $s \in \{1, \ldots S\}$. Let $\mathrm{acc}(B_s)$ and $\mathrm{conf}(B_s)$ be the averaged accuracy and averaged confidence of the examples in the bin B_s. The ECE can be defined as the following:

$$\mathrm{ECE} = \sum_{s=1}^{S}\frac{|B_s|}{n}\left|\mathrm{acc}\left(B_s\right) - \mathrm{conf}\left(B_s\right)\right|, \tag{6}$$

where n is the number of samples.

Baselines

We compare our methods (i) *Hatch Ens A*: ensemble HatchNets of the same architecture and (ii) *Hatch Ens B*: ensemble HatchNets of the different architectures, to (a) *Vanilla*: maximum softmax probability of single model [38], (b) *MC-Dropout*: Monte-Carlo Dropout with rate p [12], (c) *Deep Ens*: Ensembles of M networks trained independently on the entire dataset using random initialization (we set $M = 5$ in experiments below) [10].

Performance of HatchEnsemble under clean datasets

In this part, we focus on Question 1. In the two methods we propose, the model architectures are broadened according to the following rules:

(i) LeNet: (a) *Hatch Ens A(5)*: the number of channels in the first two layers of five models are all changed from {1/3-6-16} to {1/3-9-19} (1/3 represent the input channels of mnist and cifar10, respectively). (b) *Hatch Ens B(5)*: the number of channels in the first two layers of five models are changed from {1/3-6-16} to {1/3-7-17}, {1/3-8-18}, {1/3-9-19}, {1/3-10-20}, {1/3-11-21}, respectively.

(ii) VGG-11: (a) *Hatch Ens A(5)*: the number of channels in the first two layers of five models are all changed from {3-64-128} to {3-70-134}. (b) *Hatch Ens B(5)*: the number of channels in the first two layers of five models are changed from {3-64-128} to {3-66-130}, {3-68-132}, {3-70-134}, {3-72-136} and {3-74-138}, respectively.

(iii) ResNet-18: (a) *Hatch Ens A(5)*: the number of channels in the first two BasicBlocks of first block of five models are all changed from {64} to {70}. (b) *Hatch Ens B(5)*: the number of channels in the two BasicBlocks of first block of five models are changed from {64} to {66}, {68}, {70}, {72} and {74}, respectively.

The reason for marking the first two results with the best performance for each metric in Table 1 is that we do not necessarily need our proposed method to surpass Deep Ens on all tasks completely. What we expect is that the effect is comparable to Deep Ens. From the colored numbers in Table 1, we can see that the two proposed methods have good performance. On the tasks where Deep Ens achieve the best performance, the methods we proposed follow closely behind and are only slightly worse. Our proposed method can surpass it to get the best performance on the tasks where Deep Ens failed to achieve the best performance. This proves that our methods are effective.

Table 1 Comparison over MNIST, Fashion MNIST, CIFAR-10 and TinyImageNet with LeNet, VGG-11 and ResNet-18 models

Dataset	Architecture	Method	ACC ↑	NLL ↓	BS ↓	ECE ↓
MNIST	LeNet	Vanilla	0.9936	0.0218	0.0103	0.0064
		MC-Drop(0.1)	0.9932	0.0216	0.0101	0.0066
		MC-Drop(0.2)	0.9934	0.0214	0.0102	0.0065
		MC-Drop(0.5)	0.9937	0.0215	0.0102	0.0069
		Deep Ens(5)	0.9951	0.0158	0.0077	0.0055
		Hatch Ens A(5)	0.9941	0.0209	0.0095	0.0059
		Hatch Ens B(5)	0.9936	0.0221	0.0100	0.0068
FashionMNIST	LeNet	Vanilla	0.9116	0.2793	0.1373	0.0509
		MC-Drop(0.1)	0.9117	0.2904	0.1374	0.0506
		MC-Drop(0.2)	0.9116	0.2873	0.1372	0.0507
		MC-Drop(0.5)	0.9115	0.2811	0.1369	0.0504
		Deep Ens(5)	0.9266	0.2196	0.1102	0.0380
		Hatch Ens A(5)	0.9150	0.2402	0.1361	0.0420
		Hatch Ens B(5)	0.9132	0.2428	0.1370	0.0436
CIFAR10	VGG-11	Vanilla	0.9229	0.3166	0.1263	0.0457
		MC-Drop(0.1)	0.9225	0.3220	0.1273	0.0463
		MC-Drop(0.2)	0.9215	0.3188	0.1285	0.0470
		MC-Drop(0.5)	0.9225	0.3228	0.1286	0.0491
		Deep Ens(5)	0.9361	0.2308	0.1002	0.0343
		Hatch Ens A(5)	0.9339	0.2418	0.1029	0.0340
		Hatch Ens B(5)	0.9344	0.2429	0.1041	0.0352
CIFAR10	ResNet-18	Vanilla	0.9519	0.1885	0.0778	0.0280
		MC-Drop(0.1)	0.9498	0.1945	0.0820	0.0295
		MC-Drop(0.2)	0.9505	0.1906	0.0807	0.0285
		MC-Drop(0.5)	0.9520	0.1896	0.0787	0.0282
		Deep Ens(5)	0.9586	0.1447	0.0647	0.0211
		Hatch Ens A(5)	0.9561	0.1613	0.0706	0.0227
		Hatch Ens B(5)	0.9589	0.1506	0.0659	0.0209
TinyImageNet	ResNet-18	Vanilla	0.6200	1.6925	0.5155	0.0768
		MC-Drop(0.1)	0.6201	1.7179	0.5156	0.0909
		MC-Drop(0.2)	0.6148	1.7730	0.5270	0.1086
		MC-Drop(0.5)	0.6158	1.7089	0.5148	0.0926
		Deep Ens(5)	0.6769	1.3593	0.4342	0.0463
		Hatch Ens A(5)	0.6496	1.5193	0.4666	0.0574
		Hatch Ens B(5)	0.6486	1.5165	0.4670	0.0558

The red numbers represent each task's optimal value, and the blue numbers represent each task's suboptimal value

Performance of HatchEnsemble under corrupted datasets

In this part, we focus on Question 2. The current neural networks are too confident about their prediction results, proposed and confirmed in Ref. [6]. This feature can lead to two bad results. The first is that the neural network will produce a very confident result for data that it has never seen before, even if it is wrong. The second is that if the neural network is too confident about its output, it will think that everything is sure, which will lead to low quality of the estimated uncertainty that cannot be used as a basis for decision-making. Therefore, it is essential to evaluate the model's calibration metrics on out-of-distribution samples for uncertainty estimation.

Figures 6 and 7 summarize the acc, nll, bs and ece for CIFAR10-C and TinyImageNet-C in Task4 and Task5 across all 95 combinations of corruptions and intensities from Ref. [39]. We show the mean on the test set for each method and summarize the results on each intensity of shift with a box plot. Each box shows the quartiles summarizing the results across all 19 types of shift, while the error bars indicate the min and max across different shift types. A similar measurement can be found in Ref. [11]. We find that all methods improve upon the single model. But MC-Dropout is still much worse than the explicit ensemble methods. This is also the reason why a lot of work recently started to point out the problems of MC-Dropout [40]. Although it is simple and easy to use, the effect is mediocre.

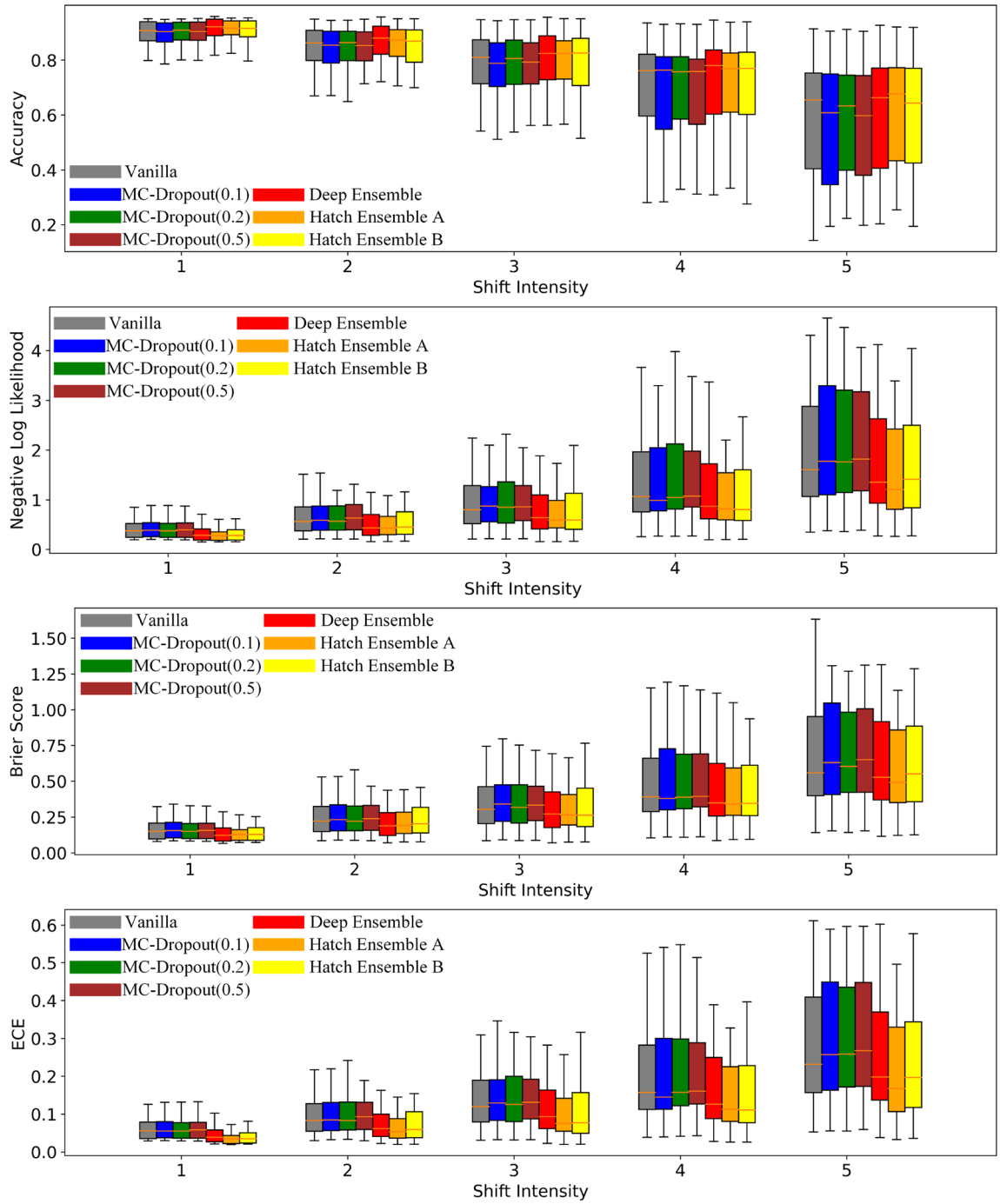


Fig. 6 Uncertainty metrics under distributional shift: a detailed comparison of accuracy, brier score, negative log likelihood and expected calibration error under all types of corruptions on **CIFAR10** with **Task4**

Comparing the three explicit ensemble methods, we find the mean of four metrics is similar for all ensemble methods, whereas the two methods we proposed show more robustness than Deep Ens as it typically leads to smaller minimums. In Fig. 6, the greater the noise intensity, the more pronounced the advantage. In Fig. 7, the advantage of the ECE is undeniable. This advantage is not only reflected in the accuracy of prediction but also the calibration of uncertainty. In the internal comparison of the two methods we proposed, Hatch Ens A is slightly better than Hatch Ens B. From another perspective, the length of our proposed method's box diagrams is shorter, reflecting that our ensemble method is not so sensitive to various types of noise and has good robustness.

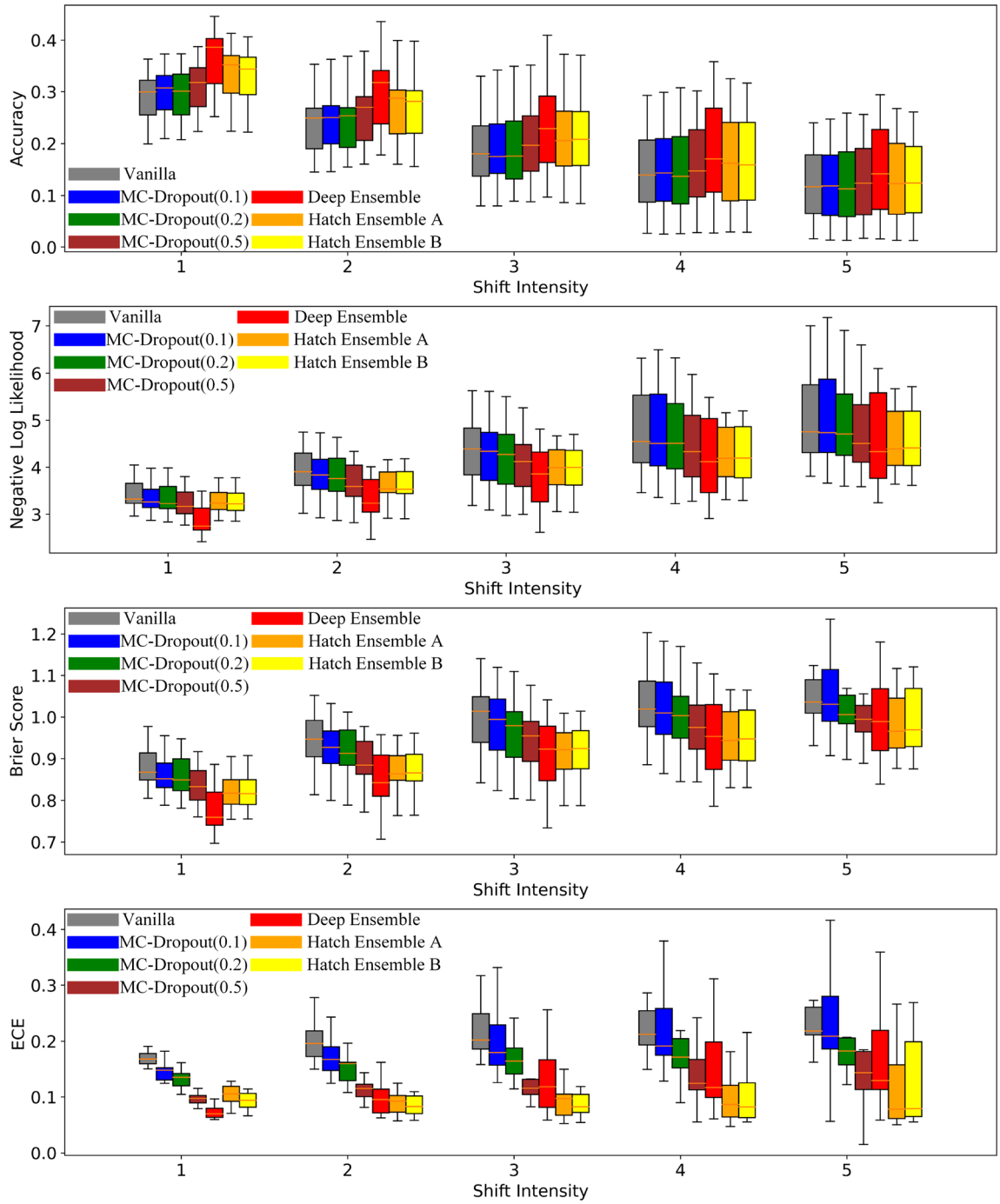
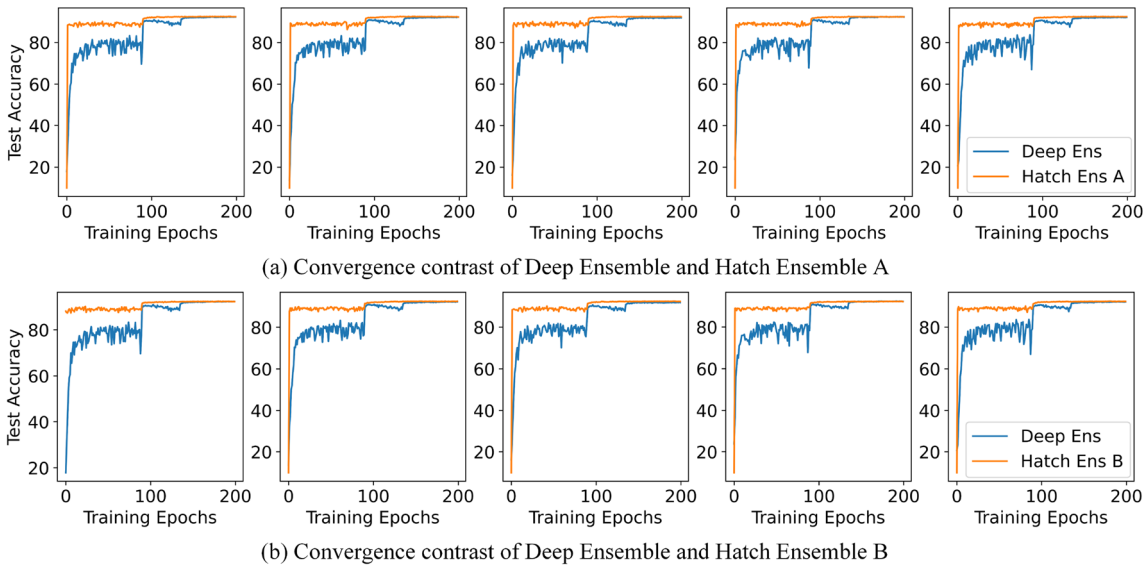


Fig. 7 Uncertainty metrics under distributional shift: a detailed comparison of accuracy, brier score, negative log likelihood and expected calibration error under all types of corruptions on **TinyImageNet** with **Task5**

Table 2 Computational costs on CIFAR10 on VGG-11 and ResNet-18

	Vanilla	Deep Ens	Hatch Ens A	Hatch Ens B
VGG-11(size=5)	1	5.25	**3.7**	3.79
ResNet-18(size=5)	1	5.24	4.17	**4.05**
ResNet-18(size=10)	1	10.15	7.41	**6.97**

Numbers under each method are relative to vanilla neural network. Numbers in () are the ensemble size



(a) Convergence contrast of Deep Ensemble and Hatch Ensemble A

(b) Convergence contrast of Deep Ensemble and Hatch Ensemble B

Fig. 8 Convergence contrast of Deep Ensemble and Hatch Ensemble. Each curve represents the convergence of the corresponding model in the corresponding method. The ensmeble size here is 5

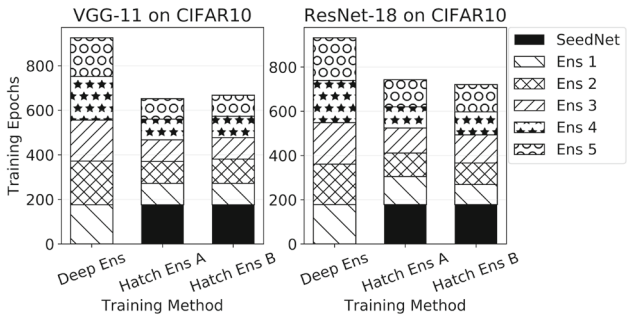


Fig. 9 HatchEnsemble trains ensemble networks significantly faster after having trained the SeedNet (shown in black)

Reducing training time cost

In this part, we focus on Question 3. In order to analyze the convergence of hatching more intuitively, we have drawn the convergence curve of the two methods when the ensemble size is 5. As shown in Fig. 8, because our method has learned the prior knowledge of SeedNet, when HatchNets are widened and then trained again, the convergence speed will be much faster. Moreover, they will reach the convergence value of each model in the standard Deep Ensemble earlier. This is the main reason for the high efficiency of HatchEnsemble.

Whether it is Deep Ens or Hatch Ens, all time consumption is spent on training the model. The size of the model is almost the same, and there are no additional algorithms, so the time consumed by each epoch of the two methods is the same. So we can equate the training epochs to time consumption. To better understand the time cost of the entire training process and how our method saves time, Fig. 9 provides the time breakdown per ensemble network. Because the experimental environment is the same, we count the number of training epochs instead of directly counting the training time. We show this with ensembling of VGG-11 and ResNet-18 on CIFAR10 and compare Hatch Ens with individual training approaches Deep Ens. While other approaches spend significant time training each network, Hatch Ens can train these networks very quickly after having trained the core SeedNet (the black part in the stacked bar in Fig. 9). Although our method needs to train one more model, it generally takes less time. We observe a similar time breakdown across all tasks in our experiments.

Specifically, for Hatch Ens, when the test accuracy on the validation set reaches the level of Deep Ens, we stop training and record the epoch at this time. We find that our proposed method reduces the time required to achieve the same effect as Deep Ens. Moreover, in the performance evaluation of "Performance of HatchEnsemble under clean datasets" and "Performance of HatchEnsemble under corrupted datasets" sections, our method is the same as Deep Ens and even better than it on some tasks. The combination of the two shows that our method has advantages in time and does not decrease in performance. Figure 9 shows that the two methods we proposed reduce about a complete training cycle in our experimental setting. From another perspective, our method trains six models faster than training five individual models by one entire training cycle. As shown in Table 2, we use the multiple relationship to show the advantage in time cost. And with the increase in ensemble size, the advantage in time cost will be magnified. This means that the more expensive the Deep Ensemble is, the more obvious the efficiency of HatchEnsemble will be improved.

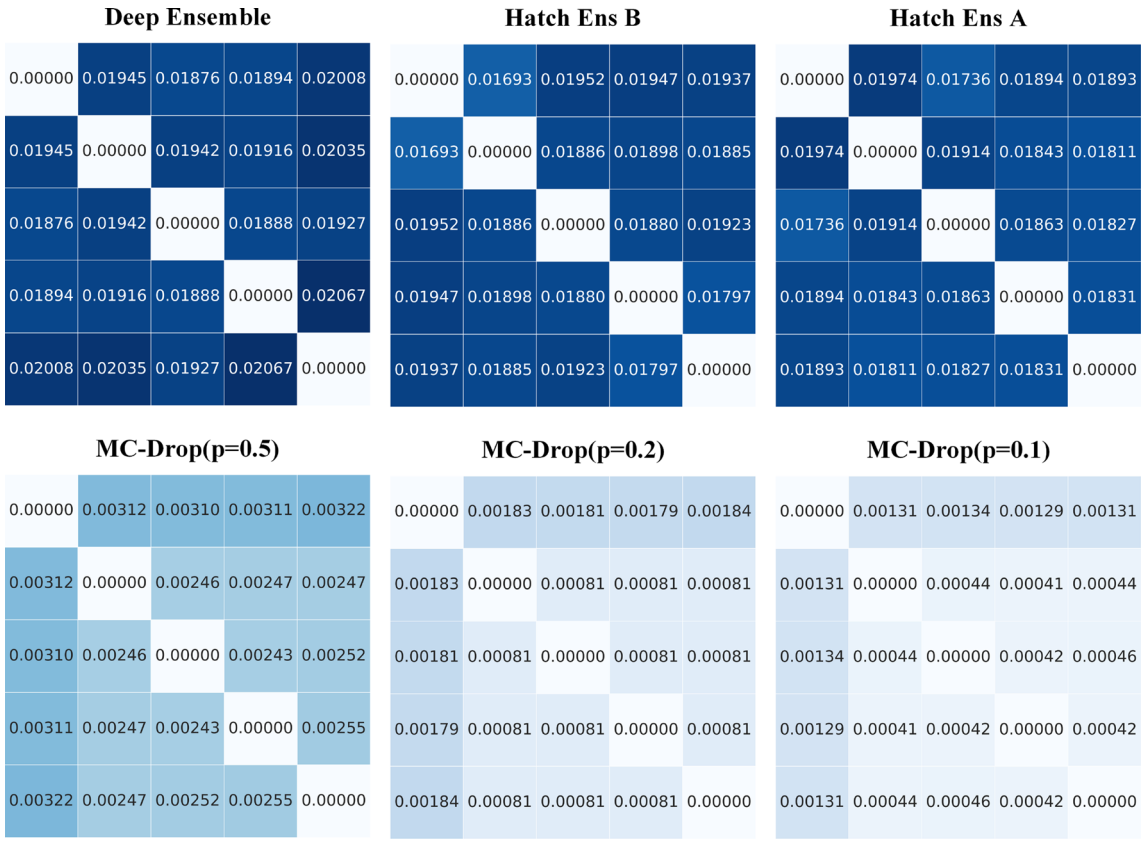


Fig. 10 Using Jensen–Shannon Divergence (JSD) to characterize the diversity of ResNet-18 models on CIFAR10 under the four methods

Diversity of model predictions

In this part, we focus on Question 4. We analyze how HatchEnsemble produces diverse ensembles compared with Deep Ensemble [10] and MC-Dropout [12].

Our goal is to observe how different training processes affect the degree of correlation between each ensemble model member. MC-Dropout can be seen as an implicit ensemble method here. To do this, we train each of the five models in Task4 under Deep Ens, MC-Dropout, Hatch Ens A, and Hatch Ens B. Letting Y_{ij} be the softmax output of the correct model on test sample j using model i, so we can think of it as a probability distribution, we then estimate Jensen-Shannon Divergence (JSD) between Y_{ij} and $Y_{i'j}$ for each i, i' and j. To get an average value for a model, instead of one for each test example, we then average across all test examples, i.e.

$$\mathrm{JSD}\left(Y_i, Y_{i'}\right) = \frac{1}{n}\sum_{j=1}^{n}\mathrm{JSD}\left(Y_{ij}, Y_{i'j}\right) \tag{7}$$

where $\mathrm{JSD}\left(Y_{ij}, Y_{i'j}\right)$ can be specifically defined as

$$\mathrm{JSD}(Y_{ij}\|Y_{i'j}) = \frac{1}{2}\mathrm{KL}(Y_{ij}\|\frac{Y_{ij}+Y_{i'j}}{2}) + \frac{1}{2}\mathrm{KL}(Y_{i'j}\|\frac{Y_{ij}+Y_{i'j}}{2}) \tag{8}$$

The reason why not directly choose to use KL divergence to measure the distance between distributions is that KL divergence is not symmetrical, resulting in two different values for the same two models. So we choose its variant Jensen–Shannon Divergence to measure the diversity between models.

Figure 10 shows the results. The value corresponding to the ith row and i'th column in each picture means the JSD of model i and model i'. Because JSD is symmetrical, the matrices in the figure are all symmetrical. Their Mean Jensen–Shannon Divergences (Mean-JSD) and Max Jensen–Shannon Divergences (Max-JSD) are shown in Table 3. The formula of Mean-JSD and Max-JSD are defined as follows:

$$\text{Mean-JSD} = \frac{1}{10}\sum_{i=1}^{4}\sum_{i'=i+1}^{5}\mathrm{JSD}\left(Y_i, Y_{i'}\right) \tag{9}$$

$$\text{Max-JSD} = \mathrm{Max}\left(\mathrm{JSD}\left(Y_i, Y_{i'}\right)\right) \tag{10}$$

As shown in Table 3, the diversity of Deep Ens is the best because the value in its matrix is the largest, followed by Hatch Ens B, then Hatch Ens A, and finally MC-Dropout. In MC-Dropout, the bigger the value of p, the greater the

Table 3 Using Mean Jensen–Shannon Divergence (Mean-JSD) and Max Jensen–Shannon Divergence (Max-JSD) to characterize the diversity of models under the four methods in Fig. 10

	Mean-JSD	Max-JSD
Deep Ensemble	0.01950	0.02067
Hatch Ens B	0.01880	0.01952
Hatch Ens A	0.01859	0.01974
MC-Dropout(p=0.5)	0.00275	0.00322
MC-Dropout(p=0.2)	0.00121	0.00184
MC-Dropout(p=0.1)	0.00078	0.00134

diversity between models. Although the method we proposed does not surpass Deep Ens in terms of diversity, they are not much different, and Hatch Ens A and Hatch Ens B are only slightly behind it. Combining the performance and time cost mentioned in the previous section, our method can achieve a good performance (equal or exceed) in a shorter period of time, and the diversity of models does not decrease much. On the whole, the practicality of our method exceeds the baseline.

The influence of ensemble size on prediction performance and uncertainty quality

In this part, we focus on Question 5. To get the influence of ensemble size on prediction performance and uncertainty quality, we change the ensemble size from 1 to 10 in Task4.

In the two methods we propose, the ResNet-18 architectures are broadened according to the following rules: (a) *Ours A*: the number of channels in the first two BasicBlocks of first block of five models are all changed from {64} to {70}. (b) *Ours B*: the number of channels in the two BasicBlocks of first block of five models are changed from {64} to {65}-{74}. If we want to ensemble M models, then take the first M from this model sequence for testing.

It can be seen from Table 4 that with the increase in the number of an ensemble, the accuracy and the quality of predictive uncertainty of the three methods have significantly improved. Lobacheva et al. [41] interpret this phenomenon as the power law in deep ensemble. Although our methods are still slightly worse than Deep Ens, they can compensate for this disadvantage by ensembling more models than Deep Ens without requiring more time consumption. From Table 2 we can calculate that the results marked with the same color in Table 4 take the same time. For example, the time cost of training 4 models by Deep Ens, 5 models by Hatch Ens A and 6 models by Hatch Ens B is the same and the result obtained by ensembling 6 models under Hatch Ens B is better than the result obtained by ensembling 4 models under Deep Ens.

In general, when the application requirements are high efficiency, our method can be well applied; when the application requirements are high performance, our method can also achieve the goal by ensembling more models in the same time as Deep Ensemble.

Conclusions and future work

We proposed an ensemble method named HatchEnsemble for quantifying uncertainty in deep neural networks. Our method can quantify the uncertainty with good quality more efficiently compared with existing non-Bayesian ensemble methods. The core intuition behind HatchEnsemble is to reduce the number of epochs needed to train an ensemble by using the knowledge learned by SeedNet and training for it once. Through comprehensive experiments, we demonstrate that HatchEnsemble can give competitive predictive accuracy with well-calibrated uncertainty in a shorter time compared with Deep Ensemble.

There are several avenues for future work. One of them is how to use NAS technology to search for possible hatch methods automatically. Diversity is another problem worthy of being studied in ensemble learning, for it is strongly

Table 4 Results on ResNet-18 over CIFAR-10: Three ensemble methods lead to higher classification accuracy and better predictive uncertainty as evidenced by lower NLL, BS and ECE during the ensemble size M increasing

	ACC↑			NLL↓			BS↓			ECE↓		
M	Deep Ens	Ours A	Ours B	Deep Ens	Ours A	Ours B	Deep Ens	Ours A	Ours B	Deep Ens	Ours A	Ours B
1	0.9519	0.9470	0.9530	0.1885	0.2068	0.1863	0.0778	0.0867	0.0769	0.0279	0.0305	0.0278
2	0.9575	0.9527	0.9571	0.1625	0.1743	0.1590	0.0695	0.0751	0.0670	0.0226	0.0249	0.0229
3	0.9581	0.9540	0.9590	0.1511	0.1663	0.1516	0.0667	0.0728	0.0650	0.0225	0.0245	0.0212
4	0.9595	0.9541	0.9604	0.1474	0.1629	0.1482	0.0651	0.0713	0.0644	0.0216	0.0246	0.0208
5	0.9586	0.9561	0.9595	0.1447	0.1613	0.1465	0.0647	0.0706	0.0640	0.0211	0.0227	0.0207
6	0.9593	0.9560	0.9601	0.1408	0.1600	0.1460	0.0630	0.0701	0.0637	0.0207	0.0227	0.0199
7	0.9594	0.9563	0.9609	0.1397	0.1602	0.1460	0.0626	0.0700	0.0639	0.0205	0.0228	0.0191
8	0.9595	0.9571	0.9601	0.1392	0.1571	0.1461	0.0629	0.0684	0.0640	0.0205	0.0223	0.0199
9	0.9603	0.9571	0.9597	0.1385	0.1576	0.1464	0.0624	0.0689	0.0641	0.0211	0.0227	0.0204
10	0.9609	0.9577	0.9600	0.1381	0.1573	0.1460	0.0619	0.0687	0.0642	0.0197	0.0219	0.0203

The numbers marked with the same color mean that it takes the same time to get the results

related to the performance of ensemble. Finally, reducing the memory costs while retaining the same performance under dataset shift would also be a key challenge.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China (No.11725211, 52005505, 6200 1502) and the Postgraduate Scientific Research Innovation Project of Hunan Province (CX20200006).

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
2. Krizhevsky A, Sutskever I, Hinton GE (2017) Imagenet classification with deep convolutional neural networks. Commun ACM 60(6):84–90
3. Hinton G, Deng L, Yu D, Dahl GE, Ar Mohamed, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN et al (2012) Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process Mag 29(6):82–97
4. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781
5. Alipanahi B, Delong A, Weirauch MT, Frey BJ (2015) Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. Nat Biotechnol 33(8):831–838
6. Guo C, Pleiss G, Sun Y, Weinberger KQ (2017) On calibration of modern neural networks. arXiv preprint arXiv:1706.0459
7. Nguyen A, Yosinski J, Clune J (2015) Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 427–436
8. Denker JS, LeCun Y (1991) Transforming neural-net output levels to probability distributions. In: Advances in neural information processing systems, pp 853–859
9. MacKay DJ (1992) A practical bayesian framework for backpropagation networks. Neural Comput 4(3):448–472
10. Lakshminarayanan B, Pritzel A, Blundell C (2017) Simple and scalable predictive uncertainty estimation using deep ensembles. Adv Neural Inf Process Syst 30:6402–6413
11. Ovadia Y, Fertig E, Ren J, Nado Z, Sculley D, Nowozin S, Dillon J, Lakshminarayanan B, Snoek J (2019) Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In: Advances in neural information processing systems, pp 13991–14002
12. Gal Y, Ghahramani Z (2016) Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: International conference on machine learning, pp 1050–1059
13. Platt J et al (1999) Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Adv Large Margin Classif 10(3):61–74
14. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
15. Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4700–4708
16. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9
17. Bernardo JM, Smith AF (2009) Bayesian theory, vol 405. Wiley, Hoboken
18. MacKay DJ (1992) Bayesian methods for adaptive models. PhD thesis, California Institute of Technology
19. Neal RM (2012) Bayesian learning for neural networks, vol 118. Springer Science & Business Media, Berlin
20. Blundell C, Cornebise J, Kavukcuoglu K, Wierstra D (2015) Weight uncertainty in neural networks. arXiv preprint arXiv:1505.05424
21. Graves A (2011) Practical variational inference for neural networks. Adv Neural Inf Process Syst 24:2348–2356
22. Li Y, Hernández-Lobato JM, Turner RE (2015) Stochastic expectation propagation. In: Advances in neural information processing systems, pp 2323–2331
23. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
24. Kingma DP, Salimans T, Welling M (2015) Variational dropout and the local reparameterization trick. In: Advances in neural information processing systems, pp 2575–2583
25. Maeda Si (2014) A bayesian encourages dropout. arXiv preprint arXiv:1412.7003
26. Gustafsson FK, Danelljan M, Schon TB (2020) Evaluating scalable bayesian deep learning methods for robust computer vision. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, pp 318–319
27. Wen Y, Tran D, Ba J (2020) Batchensemble: an alternative approach to efficient ensemble and lifelong learning. arXiv preprint arXiv:2002.06715
28. Lee S, Purushwalkam S, Cogswell M, Crandall D, Batra D (2015) Why m heads are better than one: Training a diverse ensemble of deep networks. arXiv preprint arXiv:1511.06314
29. Asif U, Tang J, Harrer S (2019) Ensemble knowledge distillation for learning improved and efficient networks. arXiv preprint arXiv:1909.08097
30. Huang G, Li Y, Pleiss G, Liu Z, Hopcroft JE, Weinberger KQ (2017) Snapshot ensembles: Train 1, get m for free. arXiv preprint arXiv:1704.00109
31. Wei T, Wang C, Rui Y, Chen CW (2016) Network morphism. In: International conference on machine learning, pp 564–572
32. Chen T, Goodfellow I, Shlens J (2015) Net2net: Accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641
33. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324

34. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556
35. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media, Berlin
36. Brier GW (1950) Verification of forecasts expressed in terms of probability. Mon Weather Rev 78(1):1–3
37. Naeini MP, Cooper G, Hauskrecht M (2015) Obtaining well calibrated probabilities using bayesian binning. In: Proceedings of the AAAI conference on artificial intelligence, vol 29
38. Hendrycks D, Gimpel K (2016) A baseline for detecting misclassified and out-of-distribution examples in neural networks. arXiv preprint arXiv:1610.02136
39. Hendrycks D, Dietterich T (2019) Benchmarking neural network robustness to common corruptions and perturbations. arXiv preprint arXiv:1903.12261
40. Verdoja F, Kyrki V (2020) Notes on the behavior of mc dropout. arXiv preprint arXiv:2008.02627
41. Lobacheva E, Chirkova N, Kodryan M, Vetrov DP (2020) On power laws in deep ensembles. Adv Neural Inf Process Syst 33:2375–2385