Neural Bootstrapper

Anonymous Author(s)
Affiliation
Address
email

Abstract

Bootstrapping has been a primary tool for ensemble and uncertainty quantification in machine learning and statistics. However, due to its nature of multiple training and resampling, bootstrapping deep neural networks is computationally burdensome; hence it has difficulties in practical application to the uncertainty estimation and related tasks. To overcome this computational bottleneck, we propose a novel approach called *Neural Bootstrapper* (NeuBoots), which learns to generate bootstrapped neural networks through single model training. NeuBoots injects the bootstrap weights into the high-level feature layers of the backbone network and outputs the bootstrapped predictions of the target, without additional parameters and the repetitive computations from scratch. We apply NeuBoots to various machine learning tasks related to uncertainty quantification, including prediction calibrations in image classification and semantic segmentation, active learning, and detection of out-of-distribution samples. Our empirical results show that NeuBoots outperforms other bagging based methods under a much lower computational cost without losing the validity of bootstrapping.

1 Introduction

Bootstrapping [5] or bagging [1] procedures have been commonly used as a primary tool in quantifying uncertainty lying on statistical inference, e.g. evaluations of standard errors, confidence intervals, and hypothetical null distribution. Despite its success in statistics and machine learning field, the naive use of bootstrap procedures in deep neural network applications has been less practical due to its computational intensity. Bootstrap procedures require evaluating a number of models; however, training multiple deep neural networks are infeasible in practice in terms of computational cost.

To utilize bootstrap for deep neural networks, we propose a novel bootstrapping procedure called **Neu**ral **Boots**trapper (NeuBoots). The proposed method is mainly motivated by Generative Bootstrap Sampler (GBS) [30], which trains a bootstrap generator by model parameterization based on Random Weight Bootstrapping (RWB, [29]) framework. For many statistical models, the idea of GBS is more theoretically valid than amortized bootstrap [25], which trains an implicit model to approximate the bootstrap distribution over model parameters. However, GBS is hardly scalable to modern deep neural networks, which have millions of parameters.

Contrary to the previous method, the proposed method is effortlessly scalable and universally applicable to the various architectures. The key idea of NeuBoots is simple; multiplying bootstrap weights to the final layer of the backbone network and instead of model parameterization. Hence it outputs the bootstrapped predictions of the target without additional parameters and the repetitive computations from scratch. NeuBoots outperforms the previous sampling-based methods [9, 18, 25] on the various uncertainty quantification related tasks with deep convolutional networks [12, 14, 16]. Throughout this paper, we show that NeuBoots has multiple advantages over the existing uncertainty quantification procedures in terms of memory efficiency and computational speed (see Table 1.1).

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

	Standard Bootstrap [5]	MCDrop [9]	Deep Ensemble [18]	NeuBoots
Memory Efficiency	✗	✓	✗	✓
Fast Training	✗	✓	✗	✓
Fast Prediction	✗	✗	✗	✓

Table 1.1. Computational comparison between bagging based uncertainty estimation methods in the view of memory efficiency and computational speed during the training and prediction step.

To verify the empirical power of the proposed method, we apply NeuBoots to a wide range of experiments related to uncertainty quantification and bagging. We apply the NeuBoots to prediction calibration, active learning, out-of-distribution (OOD) detection, semantic segmentation, and learning on imbalanced datasets. Notably, we test the proposed method on biomedical data of high-resolution, NIH3T3 data [3]. In Section 4, our results show that NeuBoots achieves at least comparable or better performance than the state-of-the-art methods in the considered applications.

2 Preliminaries

As preliminaries, we briefly review the standard bootstrapping [5] and introduce an idea of Generative Bootstrap Sampler (GBS, [30]), which is the primary motivation of the proposed method. Let $[m] := \{1, \dots, m\}$ and denote a given training data by $\mathcal{D} = \{(X_i, y_i) : i \in [n]\}$, where each feature $X_i \in \mathcal{X} \subset \mathbb{R}^p$ and its response $y_i \in \mathbb{R}^d$. We denote the class of models $f : \mathbb{R}^p \to \mathbb{R}^d$ by $\mathcal{M}$. For the standard bootstrapping, we sample B sets of bootstrap data $\mathcal{D}^{(b)} = \{(X_i^{(b)}, y_i^{(b)}) : i \in [n]\}$ with replacement for $b \in [B]$. For each bootstrap data $\mathcal{D}^{(b)}$, we define a loss functional L on $f \in \mathcal{M}$:

$$L(f, \mathcal{D}^{(b)}) := \frac{1}{n} \sum_{i=1}^{n} \ell(f(X_i^{(b)}), y_i^{(b)}) \tag{2.1}$$

where $\ell : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is an arbitrary loss function. Then we minimize (2.1) with respect to $f \in \mathcal{M}$ to obtain bootstrapped models: for $b \in [B]$,

$$\widehat{f}^{(b)} = \arg\min_{f \in \mathcal{M}} L(f, \mathcal{D}^{(b)}). \tag{2.2}$$

Random Weight Bootstrapping It is well-known that the standard bootstrap uses only (approximately) 63% of observations for each bootstrap evaluation [18]. To resolve this problem, we use Random Weight Bootstrapping (RWB, [29]), which reformulates (2.2) as a sampling of bootstrapping weights for a weighted loss functional. Let $\mathcal{W} = \{\mathbf{w} \in \mathbb{R}_+^n : \sum_{i=1}^n w_i = n\}$ be a dilated standard $(n-1)$-simplex. For $\mathbf{w} = (w_1, \dots, w_n) \in \mathcal{W}$ and the original training data $\mathcal{D}$, we define the Weighted Bootstrapping Loss (WBL) functional on $f \in \mathcal{M}$ as follows:

$$L(f, \mathbf{w}, \mathcal{D}) := \frac{1}{n} \sum_{i=1}^{n} w_i \ell(f(X_i), y_i). \tag{2.3}$$

Then for any resampled dataset $\mathcal{D}^{(b)}$, there exists a unique $\mathbf{w} \in \mathcal{W}$ such that (2.1) matches to (2.3). This reformulation provides a relaxation method to consider full data set without any omission in bootstrapping. Precisely, as a continuous relaxation of the standard bootstrap, we use Dirichlet distribution [26]; $\mathbb{P}_{\mathcal{W}} = n \times \text{Dirichlet}(1, \dots, 1)$, where $\mathbb{P}_{\mathcal{W}}$ is a probability distribution on the simplex $\mathcal{W}$. Hence RWB fully utilizes the observed data points, since sampled bootstrap weights $\mathbf{w} \sim \mathbb{P}_{\mathcal{W}}$ are strictly positive. Also, [27] showed that RWB achieves the same theoretical properties with these of the standard bootstrap i.e. $\mathbb{P}_{\mathcal{W}} = \text{Multinomial}(n; 1/n, \dots, 1/n)$ in (2.3).

Bootstrap Distribution Generator Although RWB resolves the data discard problem, training multiple networks $\widehat{f}^{(1)}, \dots, \widehat{f}^{(B)}$ remains a computational problem, and one has to store the parameters of every network for prediction. To reduce the computational bottlenecks, GBS [30] proposes a procedure to train a generator function of bootstrapped estimators for parametric statistical models. The main idea of GBS is to parameterize the model parameter with bootstrap weight $\mathbf{w} \in \mathcal{W}$. When the GBS is applied to bootstrapping neural networks, it considers a *bootstrap generator*

$g : \mathbb{R}^p \times \mathcal{W} \to \mathbb{R}^d$ with parameter $\theta(\mathbf{w})$, where d is the total number of neural net parameters in g, so that $g(X, \mathbf{w}) = g_{\theta(\mathbf{w})}(X)$. Based on (2.3), we define a new WBL functional:

$$\mathcal{L}(g, \mathcal{D}) = \mathbb{E}_{\mathbf{w} \sim \mathbb{P}_{\mathcal{W}}}[L(g, \mathbf{w}, \mathcal{D})], \quad L(g, \mathbf{w}, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} w_i \ell(g(X_i, \mathbf{w}), y_i) \tag{2.4}$$

Note that we use the Dirichlet distribution for $\mathbb{P}_{\mathcal{W}}$; hence the functional $\mathcal{L}(g, \mathcal{D})$ includes RWB procedure itself. Analogous to (2.2), we obtain the bootstrap generator $\widehat{g}$ by optimizing $\mathcal{L}(g, \mathcal{D})$:

$$\widehat{g} = \arg\min_{g \in \mathcal{M}} \mathcal{L}(g, \mathcal{D}) \tag{2.5}$$

Then learned $\widehat{g}$ can generate bootstrap samples for given target data X_* by plugging an arbitrary $\mathbf{w} \in \mathcal{W}$ into $\widehat{g}(X_*, \cdot)$. We refer to [30, Section 2] for detailed theoretical results on GBS.

Block Bootstrapping The above bootstrap generator g receives a bootstrap weight vector $\mathbf{w}$ of dimension n; hence its optimization via (2.5) would be hurdled when the number of data n is large. Hence we utilize a block bootstrapping procedure to reduce the dimension of bootstrap weight vector. We allocate the index set $[n]$ to S number of blocks. Let $u : [n] \to [S]$ denotes the assignment function. Then we impose the same value of weight on all elements in a block such as, $w_i = \alpha_s$ for $u(i) = s \in [S]$, where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_S) \sim S \times \text{Dirichlet}(1, \ldots, 1)$. Instead of $\mathbf{w}$, we plug $\boldsymbol{\alpha}$ in $g(X, \cdot) = g_{\theta(\cdot)}(X)$ to generate bootstrap samples and compute the weighted loss function in (2.4):

$$\mathcal{L}(g, \mathcal{D}) = \mathbb{E}_{\boldsymbol{\alpha} \sim S \times \text{Dirichlet}(1, \ldots, 1)} \left[\frac{1}{n} \sum_{i=1}^{n} \alpha_{u(i)} \ell(g(X_i, \boldsymbol{\alpha}), y_i) \right] \tag{2.6}$$

The above procedure asymptotically converges to the same target distribution where the conventional non-block bootstrap converges. See appendix A for more detailed procedure and proofs.

3 Neural Bootstrapper

Now we propose **Neu**ral **Boots**trapper (NeuBoots), which reduces computational complexity and memory requirement of the networks in the learning of bootstrapped distribution to being suitable for deep neural networks.

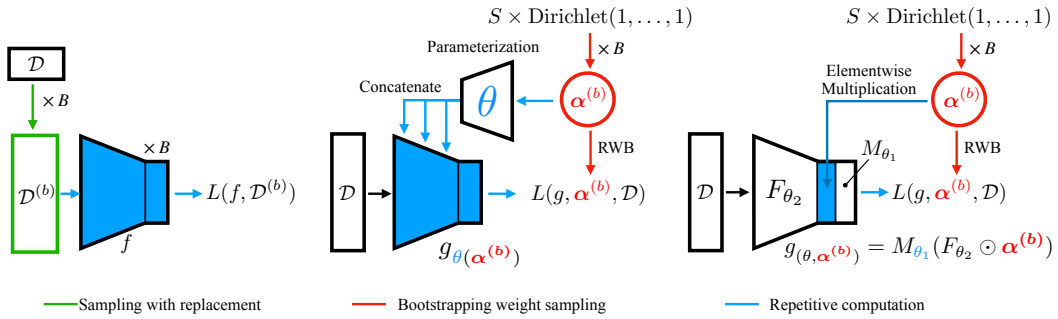
How to implement the bootstrap generator g for deep neural networks? One may consider directly applying GBS to existing deep neural networks by modeling a neural net $\theta(\cdot)$ that outputs the neural net parameters of g. However, this approach is computationally challenging due to the high-dimensionality of the output dimension of $\theta(\cdot)$ Indeed, [30] proposes an architecture which concatenates bootstrap weight vector to every layer of a given neural network (Figure 3.1(b)) and trains it with (2.6). However, the bagging performance of GBS gradually degrades as we applied it to the deeper neural networks. This may be because the information of bootstrap weights in the earlier layers less propagate since the target model reduces the parameters of the weights during the training.

3.1 Adaptive Block Bootstrapping

We found that the bootstrap weight in the final layer mainly affects the bootstrap performance of GBS. This fact motivates us to utilize the following adaptive block bootstrapping, which is the key idea of NeuBoots. Take a neural network $f_\theta \in \mathcal{M}$ with parameter θ. Let M_{θ_1} and F_{θ_2} be the single-layer neural network in the final layer and the feature extractor of f, respectively, with parameter $\theta = (\theta_1, \theta_2)$, so we can decompose f_θ into $M_{\theta_1} \circ F_{\theta_2}$. Set $S := \dim(F_{\theta_2}(X))$ for the number of blocks for block bootstrapping. Then, we redefine bootstrap generator as follows:

$$g_\theta(X, \boldsymbol{\alpha}) := g_{(\theta, \boldsymbol{\alpha})}(X) = M_{\theta_1}(F_{\theta_2}(X) \odot \boldsymbol{\alpha}) \tag{3.1}$$

where $\odot$ denotes an elementwise multiplication. Bootstrap generator (3.1) can also be trained with (2.6); hence optimized $\widehat{g}_\theta(X, \cdot)$ can generate the bootstrapped prediction as we plug $\boldsymbol{\alpha}$. This modification brings a computational benefit, since we can generate bootstrap samples quickly and memory-efficiently by reusing a priori computed tensor $F_{\theta_2}(X)$ without repetitive computation from scratch. See Figure 3.1 for the comparison between the previous methods and NeuBoots.



(a) Standard Bootstrapping (b) GBS (c) NeuBoots

Figure. 3.1. A comparison between the bootstrapping procedure of (a) standard bootstrapping [5], (b) GBS [30], and (c) NeuBoots. This figure is best viewed in color.

Training step At every epoch, we update the $\mathbf{w}_{\boldsymbol{\alpha}} = \{\alpha_{u(1)}, \ldots, \alpha_{u(n)}\}$ randomly, and the expectation in (2.6) can be approximated by the average over the sampled weights. Considering the stochastic gradient descent (SGD) algorithms to update the parameter θ via mini-batch sequence $\{\mathcal{D}_k : \mathcal{D}_k \subset \mathcal{D}\}_{k=1}^{K}$, we plug mini-batch size of bootstrap weight vector $\{\alpha_{u(i)} : X_i \in \mathcal{D}_k\}$ in (2.6) without changing $\boldsymbol{\alpha}$. Each element of $w_{\boldsymbol{\alpha}}$ is not used repeatedly during the epoch, so the sampling and replacement procedures in Algorithm 1 are conducted once at the beginning of epoch.

Algorithm 1: Training step in NeuBoots.

Input : Dataset $\mathcal{D}$; epochs T; dimension of feature S; index function u; learning rate ρ.

Initialize neural network parameter $\phi^{(0)}$ and set $n := |\mathcal{D}|$.

for $t \in \{0, \ldots, T-1\}$ **do**

  Sample $\boldsymbol{\alpha}^{(t)} = \{\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\} \overset{\text{i.i.d.}}{\sim} S \times \text{Dirichlet}(1, \ldots, 1)$

  Replace $\mathbf{w}_{\boldsymbol{\alpha}}^{(t)} = \{\alpha_{u(1)}^{(t)}, \ldots, \alpha_{u(n)}^{(t)}\}$

  Update $\theta^{(t+1)} \leftarrow \theta^{(t)} - \rho \nabla_\theta L(g_{(\theta, \boldsymbol{\alpha})}, \mathbf{w}_{\boldsymbol{\alpha}}^{(t)}, \mathcal{D})\big|_{\theta=\theta^{(t)}}$.

Algorithm 2: Prediction step in NeuBoots.

Input : Data point $X_* \in \mathbb{R}^p$; number of bootstrap sampling B.

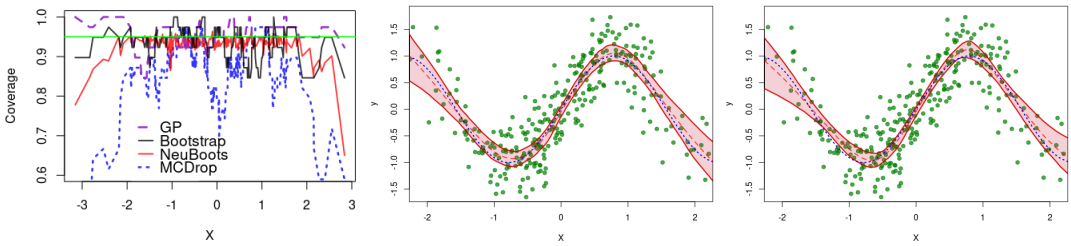
Compute the feed-forward network $\widehat{g}_*(\cdot) = \widehat{g}_\theta(X_*, \cdot)$ a priori.

for $b \in \{1, \ldots, B\}$ **do**

  Generate $\boldsymbol{\alpha}^{(b)} \overset{\text{i.i.d.}}{\sim} S \times \text{Dirichlet}(1, \ldots, 1)$ and evaluate $\widehat{y}_*^{(b)} = \widehat{g}_*(\boldsymbol{\alpha}^{(b)})$.

3.2 Discussion

NeuBoots vs Standard Bootstrap To examine the approximation power of NeuBoots, we have measured the frequentist's coverage rate of the confidence bands (Figure 3.2.(a)). We estimate 95% confidence band for nonparametric regression function by using the NeuBoots, and compare it with credible bands (or confidence bands) evaluated by the standard bootstrap, Gaussian Process (GP) regression, and MCDrop [9]. We adopt Algorithm 1 to train the NeuBoots generator with 3 hidden-layers with 500 hidden-nodes for each layer. For the standard bootstrap, we train 1000 neural networks. The result shows the confidence band via NeuBoots stably covers the true regression function on each predictor value with almost 95% of frequency, which is compatible with the standard bootstrapping. In contrast, the coverage of the MCDrop is unstable and sometimes below 70%. This result indicates that the NeuBoots performs comparably with the standard bootstrapping in uncertainty quantification tasks.



(a) Frequentist Coverage **(b)** NeuBoots **(c)** Standard Bootstrap

Figure. 3.2. (a) Frequentist coverage rate of the 95% confidence bands; (b) Curve fitting with different nonlinear funtions. 95% confidence band of the regression mean from NeuBoots; (c) the standard bootstrap. Each red dashed line indicates the mean, and the blue dotted lines show the true regression function.

NeuBoots vs Amortized Bootstrapping We applied NeuBoots to classification and regression experiments presented by the amortized bootstrap [25]. Indeed, every experiment demonstrates that NeuBoots outperforms the amortized bootstrap in bagging performance for various tasks: the rotated MNIST classification (Table 3.1), classification with different data points N (Figure B.1), and regression on two datasets (Figure B.2). We remark the expected calibration error (ECE, [24]) score on the rotated MNIST is improved via NeuBoots from 15.00 to 2.98 by increasing the number of bootstrap sampling B.

Methods	Test error for ensemble		
	$B = 1$	$B = 5$	$B = 25$
Traditional Bootstrap	22.57	19.68	18.57
Amortized Bootstrap [25]	**17.03**	16.82	16.18
NeuBoots	17.94±0.74	**14.98**±0.31	**14.45**±0.31

Table 3.1. Rotated MNIST classification with different bootstrap sampling number B.

NeuBoots vs Dropout At first glance, NeuBoots is similar to Dropout except for RWB. However, random weights imposed by the Dropout are lack of connection to the loss function nor the working model, while the bootstrap weights of the NeuBoots appears in the loss function (2.6) have explicit connections to the bootstrapping model. We briefly verify the effect of the loss function on the 3-layers MLP with the different number of hidden variables 50, 100, and 200 for the image classification task on MNIST datasets. With batch normalization [15], we have applied Dropout with probability $p = 0.1$ only to the final layer of MLP. We measure the ECE, the negative log-likelihood (NLL), and the Brier score for comparisons. NeuBoots and Dropout records same accuracy. However, Figure B.3 shows that the NeuBoots is more feasible for confidence-aware learning and clearly outperforms the Dropout in terms of ECE, NLL, and the Brier score.

Computation time After we obtain the optimized network $\widehat{g}_\theta$, for the prediction, we use the generator $\widehat{g}_*(\cdot) = \widehat{g}_\theta(X_*, \cdot)$ for a given data point X_*. Then we can generate bootstrapped predictions by plugging $\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(B)}$ in the generator $\widehat{g}_*(\cdot)$, as described in Algorithm 2. As we mentioned earlier, the algorithm evaluates the network from scratch for only once to store the tensor $F_{\theta_2}(X_*)$, while the standard bootstrapping and MCDrop [9] need repetitive feed-forward propagations. To check this empirically, we measure the prediction time by ResNet-34 between NeuBoots and MCDrop on the test set of CIFAR-10 with Nvidia V100 GPUs. NeuBoots predicts $B = 100$ bootstrapping in 1.9s whereas MCDrop takes 112s to generate 100 outputs. Also, NeuBoots is computationally more efficient than the standard bootstrap and the sparse GP [32] (Figure 3.3). For more comparison, see Table 4.1 and Table 4.3 with various architectures and tasks.

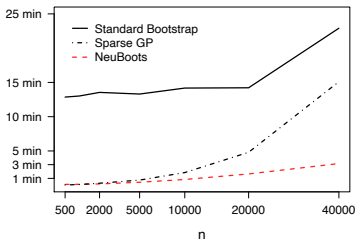


Figure. 3.3. Comparison of computational time with different numbers of training data n for the example in Figure 3.2

4 Empirical Studies

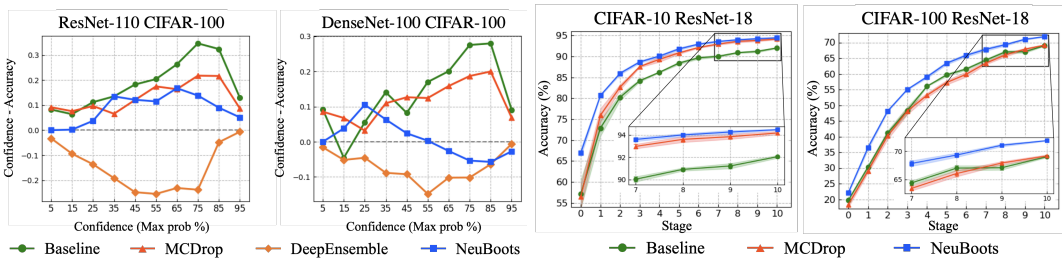
In this section, we conduct the wide range of empirical studies of NeuBoots for uncertainty quantification and bagging performance. We apply NeuBoots to prediction calibration, active learning, out-of-distribution detection, bagging performance for semantic segmentation, and learning on imbalanced dataset with various deep convolutional neural networks. Our code is open to the public[1].

4.1 Prediction Calibration

Setting We evaluate the proposed method on the prediction calibration for image classification tasks. We apply NeuBoots to image classification tasks on CIFAR and SVHN with ResNet-110 and DenseNet-100. We take $k = 5$ predictions of MCDrop and DeepEnsemble for calibration. For fair comparisons, we set the number of bootstrap sampling $B = 5$ as well, and fix the other hyperparameters same with baseline models. All models are trained using SGD with a momentum of 0.9, an initial learning rate of 0.1, and a weight decay of 0.0005 with the mini-batch size of 128. We use CosineAnnealing for the learning rate scheduler. We implement MCDrop and evaluates its performance with dropout rate $p = 0.2$, which is a close setting to the original paper. For Deep Ensemble, we utilize adversarial training and the Brier loss function. For the metric, we evaluate the error rate, ECE, NLL, and Brier score. We also compute each method's training and prediction times to compare the relative speed based on the baseline method.

Architecture	Method	Relative Training Time	Relative Prediction Time	Error Rate(%)	ECE(%)	NLL(%)	Brier Score(%)
ResNet-110	Baseline	1.0	1.0	26.69±0.35	16.43±0.15	14.19±0.71	42.09±0.51
	MCDrop	1.1	5.0	**26.45**±0.08	13.65±1.25	**13.16**±0.64	40.46±0.30
	DeepEnsemble	9.5	5.0	34.84±0.21	27.33±4.92	18.69±1.44	56.12±3.02
	NeuBoots	**1.1**	**1.2**	26.53±0.19	**8.13**±0.28	15.68±0.31	**39.31**±0.64
DenseNet-100	Baseline	1.0	1.0	24.02±0.3	12.38±0.21	10.93±0.34	36.40±0.63
	MCDrop	1.1	5.0	23.88±0.09	9.49±0.35	10.22±0.86	34.94±0.67
	DeepEnsemble	9.5	5.0	25.51±0.24	6.67±5.06	**9.66**±0.24	35.33±1.21
	NeuBoots	**1.1**	**1.3**	**23.46**±0.09	**2.38**±0.12	11.58±0.13	**34.67**±0.24

Table 4.1. Comparison of the relative training speed, relative prediction speed, error rate, ECE, NLL, and Brier on CIFAR-100. For each metric, the lower value means the better. Relative training and relative prediction times are normalized with respect to the baseline method. Best results are indicated in bold.



(a) Reliability Diagram **(b)** Active Learning

Figure. 4.1. (a) Comparison of reliability diagrams for ResNet-110 and DenseNet-100 on CIFAR-100. Confidence is the value of the maximal softmax output. A dashed black line represents a perfectly calibrated prediction. Points below this line indicate to under-confident predictions, whereas points above the line mean overconfident predictions. (b) Actice learning performance on CIFAR-10 (left) and CIFAR-100 (right) with Random, MCDrop, and NeuBoots. Curves are averaged over five runs and shaded regions denote the confidence intervals.

Results Table 4.1 validates that the NeuBoots generally show a competitive calibration ability compared to MCDrop and Deep Ensemble on CIFAR-100. See Table B.1 for the other data. Figure 4.1.(a) shows the reliability diagrams of ResNet-110 and DenseNet-100 on CIFAR-100. We observe that NeuBoots secures accuracy and prediction calibration in the image classification tasks with ResNet-110 and DenseNet-100. NeuBoots is faster in prediction than MCDrop and Deep Ensemble

[1] https://anonymous.4open.science/r/NeuBoots-7161

at least three times. Furthermore, NeuBoots shows faster in training than Deep Ensemble at least nine times. This gap increases as the number of predictions k increases. It concludes that NeuBoots outperforms MCDrop and Deep Ensemble in calibrating the prediction with the relatively faster prediction time.

4.2 Active Learning

Setting We evaluate the NeuBoots on the active learning with ResNet-18 architecture on CIFAR. For a comparison, we consider MCDrop with entropy-based sampling and random sampling. We follow an ordinary process to evaluate the performance of active learning (see [23]). Initially, a randomly sampled 2,000 labeled images are given, and we train a model. Based on the uncertainty estimation of each model, we sample 2,000 additional images from the unlabeled dataset and add to the labeled dataset for the next stage. We continue this process ten times for a single trial and repeat five trials for each model.

Results Figure 4.1.(b) shows the sequential performance improvement on CIFAR-10 and CIFAR-100. Note that CIFAR-100 is more challenging dataset than CIFAR-10. Both plots demonstrate that NeuBoots is superior to the other sampling methods in the active learning task. NeuBoots records 71.6% accuracy in CIFAR-100 and 2.5% gap with MCDrop. Through the experiment, we verify that NeuBoots has a significant advantage in active learning.

4.3 Out-of-Distribution Detection

Setting As an important application of uncertainty quantification, we have applied NeuBoots to detection of out-of-distribution (OOD) samples. At first, we train ResNet-34 for the classification task in CIFAR-10 (in-distribution). We use the test datasets only for model evaluation. Then, we evaluate the performance of NeuBoots for OOD detection in the SVHN (out-of-distribution). For each model and dataset, we tune hyperparameters in the training phase using in-distribution samples to keep the fairness of our method. In the evaluation phase, we use a logistic regression based detector which outputs a confidence score for given test sample to discriminate OOD samples from in-distribution dataset. To evaluate the performance of the detector, we measure the true negative rate (TNR) at 95% true positive rate (TPR), the are under the receiver operating characteristic curve (AUROC), the area under the precision-recall curve (AUPR), and the detection accuracy. For comparison, we examine the baseline method [13], ODIN [21], and Mahalanobis [20]. For our method, we tune whole hyperparameters using a separate validation set, which consists of 1,000 images from in-distribution and out-distribution, respectively. After the training, we estimate the following four statistics regarding logit vectors: the max of predictive mean vectors, the standard deviation of logit vectors, expected entropy, and predictive entropy, which can be computed by the sampled output vectors of NeuBoots. Based on these statistics, similar to [20, 22], we tune the weights of logistic regression detector using nested cross-validation within the validation set, where the label is annotated positive for in-distribution sample and annotated negative for out-distribution sample.

Method	TNR at TPR 95%	AUROC	Detection Accuracy	AUPR In	AUPR Out
Baseline	32.47	89.88	85.06	85.4	93.96
ODIN	86.55	96.65	91.08	92.54	98.52
Mahalanobis	54.51	93.92	89.13	91.54	98.52
+ Calibration	96.42	**99.14**	95.75	**98.26**	99.6
NeuBoots	89.40	97.26	93.80	93.97	98.86
+ Calibration	**99.00**	**99.14**	**96.52**	97.78	**99.68**

Table 4.2. OOD detection. All values are percantages and the best results are indicated in bold.

Results Table 4.2 shows NeuBoots significantly outperform the baseline method [13] and ODIN [21] without any calibration technique in OOD detection. Furthermore, with the input pre-processing technique studied in [21], NeuBoots is superior to Mahalanobis [20] in most metrics, which employs both the feature ensemble and the input pre-processing for the calibration techniques. This validates NeuBoots can discriminate OOD samples effectively. In order to see the performance change of the

Dataset	Architecture	Method	mIoU(%)	ECE(%)	Relative Prediction Time
2D (PASCAL VOC [7])	ResNet-50	Baseline	84.57±0.72	15.35±0.21	1.0
		MCDrop	87.81±1.83	6.6±0.1	5.4
		NeuBoots	**90.14**±2.17	**6.00**±0.1	**2.7**
	ResNet-101	Baseline	85.35±0.23	15.49±0.44	1.0
		MCDrop	88.08±1.80	6.48±0.08	5.3
		NeuBoots	**90.56**±1.71	**6.14**±0.11	**2.5**
3D (NIH3T3 [3])	U-ResNet	Baseline	61.54±1.14	1.85±0.19	1.0
		MCDrop	64.15±0.48	**1.53**±0.09	5.5
		NeuBoots	**67.78**±1.01	1.67±0.19	**4.7**
	SCNAS [16]	Baseline	67.52±1.95	1.45±0.19	1.0
		MCDrop	65.37±1.13	0.64±0.17	5.9
		NeuBoots	**70.80**±1.58	**0.63**±0.16	**4.0**

Table 4.3. Semantic segmentation. The best results are indicated in bold.

OOD detector concerning the bootstrap sample size, we evaluate the predictive standard deviation estimated by the proposed method for different $B \in \{2, 5, 10, 20, 30\}$. Figure B.4 illustrates that the NeuBoots successfully detects the in-distribution samples (top row) and the out-of-distribution samples (bottom row).

4.4 Bagging Performance for Semantic Segmentation

Setting To demonstrate the applicability of NeuBoots to different computer vision tasks, we validate NeuBoots on PASCAL VOC 2012 semantic segmentation benchmark [7] with DeepLab-v3 [2] based on the backbone architectures of ResNet-50 and ResNet-101. We modify the final 1×1 convolution layer after the Atrous Spatial Pyramid Pooling (ASPP) module by multiplying the channel-wise bootstrap weights. This is a natural modification of the segmentation architecture analogous to the fully connected layer of the networks for classification tasks. Additionally, we apply NeuBoots to real 3D image segmentation task on commercial ODT microscopy NIH3T3 [3] dataset, which is challenging for not only models but also human due to the $512 \times 512 \times 64$ sized large resolution and endogenous cellular variability. We use two different U-Net-like models for this 3D image segmentation task, which are U-ResNet and SCNAS. We simply amend the bottleneck layer in the same way as the 2D version. Same as an image classification task, we set $B = 5$ and $k = 5$ for NeuBoots and MCDrop, respectively. For the remaining, we follow the usual setting.

Results Table 4.3 shows NeuBoots significantly improves mean IoU and ECE compared to the baseline. Furthermore, similar to the image classification task, NeuBoots records faster prediction time than MCDrop. This experiment indeed verifies that NeuBoots can be applied to the general scope of computer vision tasks beyond image classification.

4.5 Imbalanced Dataset

Setting To validate the efficacy for the imbalanced dataset, we have applied NeuBoots to two imbalance sets, the imbalanced CIFAR-10 and the white blood cell dataset with ResNet-18. To conduct an imbalanced CIFAR-10, we randomly sampled from the training dataset of CIFAR-10 to follow a different distribution for each class, and the distribution is [*50, 100, 150, 200, 250, 300, 350, 400, 450, 500*] for [*airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*]. The white blood cell dataset was acquired using a commercial ODT microscopy, and each image is a grayscale of 80×80. The dataset comprises four types of white blood cells, and the training distribution is [*144, 281, 2195, 3177*] for [*eosinophil, monocyte, lymphocyte, neutrophil*]. ResNet-18 model and MCDrop extension are used as a baseline and comparative model with the same settings as Section 4.2, respectively. We measure the F1 Score for each class for evaluation.

Result Comparing the performance of Baseline and MCDrop, NeuBoots performs better on both imbalanced CIFAR-10 and the white blood cell dataset, as shown in Figure 4.2. Especially, NeuBoots outperforms for eosinophil identification, the class with the lowest number of samples in the white

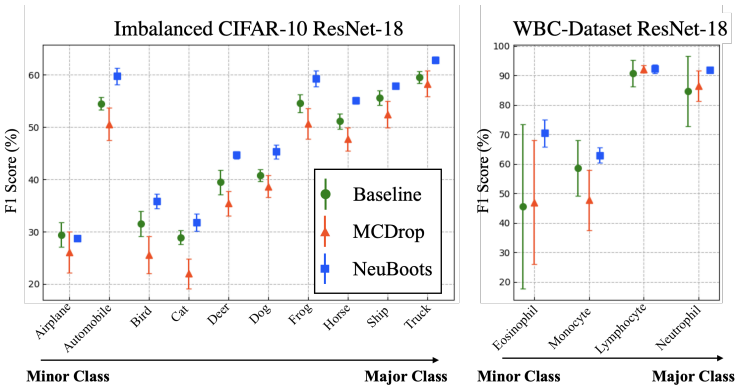


Figure. 4.2. Comparisons of classification power for imbalance datasets. The minor class refers to the class with the least number of samples, and the major class refers to the highest number of samples.

blood cell dataset, with low variance. This result shows that the NeuBoots boosts the prediction power for the fewer sampled classes with high stability via simple implementation.

5 Related Work

Bootstrapping Neural Network Since [5] first proposed the nonparametric bootstrapping to quantify uncertainty in general settings, there has been a rich amount of literature that investigate theoretical advantages of using bootstrap procedures [10], [11], and [6] showed that bootstrap procedures are capable of achieving second-order correctness. Franke and Neumann [8] investigated bootstrap consistency of one-layered MLP under some strong regularity conditions. Reed et al. [28] considered using a conventional nonparametric bootstrapping to robustify classifiers under noisy labeling. However, due to the nature of repetitive computations, its practical application to large-sized data sets is not trivial. Nalisnick and Smyth [25] proposed an approximation of bootstrapping for neural network by applying amortized variational Bayes. Despite its computational efficiency, the armortized bootstrap does not induce the exact target bootstrap distribution, and its theoretical justification is lacking. Recently, [19] proposes a bootstrapping method for neural processes. They utilized residual bootstrap to resolve the data discard problem, but their approach is not scalable since it requires multiple encoder computations.

Uncertainty Estimation There are many approaches to estimate the confidence intervals of prediction of the deep neural networks. Deep Confidence [4] proposes a framework to compute confidence intervals for individual predictions using snapshot ensembling and conformal prediction. Also, a calibration procedure to approximate a confidence interval is proposed based on Bayesain neural networks [17]. Previously, approximate inference in Bayesian neural networks has been proposed mainly in the literature. Gal and Ghahramani [9] proposes MCDrop which captures model uncertainty casting dropout training in neural networks as an approximation of variational Bayes. Smith and Gal [31] examines various measures of uncertainty for adversarial example detection. Lakshminarayanan et al. [18] proposes a non-Bayesian approach, called DeepEnsemble, to estimate predictive uncertainty based on ensembles and adversarial training. Compared to DeepEnsemble, NeuBoots does not require adversarial training nor learning multiple models.

6 Conclusion

We introduced a novel and scalable bootstrapping method, NeuBoots, for neural networks. We applied it to the wide range of machine learning tasks related to uncertainty quantification; prediction calibration, active learning, out-of-distribution detection, and imbalanced datasets. NeuBoots also demonstrates superior bagging performance over semantic segmentation. Our empirical studies show that NeuBoots attains significant potential in quantifying uncertainty for large-sized applications, such as biomedical data analysis with high-resolution. As a future research, one can apply NeuBoots to natural language processing tasks using Transformor [33].

References

[1] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

[2] Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*.

[3] Choi, J., Kim, H.-J., Sim, G., Lee, S., Park, W. S., Park, J. H., Kang, H.-Y., Lee, M., Heo, W. D., Choo, J., Min, H., and Park, Y. (2021). Label-free three-dimensional analyses of live cells with deep-learning-based segmentation exploiting refractive index distributions. *bioRxiv*.

[4] Cortés-Ciriano, I. and Bender, A. (2018). Deep confidence: a computationally efficient framework for calculating reliable prediction errors for deep neural networks. *Journal of chemical information and modeling*, 59(3):1269–1281.

[5] Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26.

[6] Efron, B. (1987). Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82(397):171–185.

[7] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338.

[8] Franke, J. and Neumann, M. H. (2000). Bootstrapping neural networks. *Neural computation*, 12(8):1929–1949.

[9] Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

[10] Hall, P. (1986). On the bootstrap and confidence intervals. *The Annals of Statistics*, pages 1431–1452.

[11] Hall, P. (1992). On bootstrap confidence intervals in nonparametric regression. *The Annals of Statistics*, pages 695–711.

[12] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[13] Hendrycks, D. and Gimpel, K. (2017). A baseline for detecting misclassified and out-of-distribution examples in neural networks. *Proceedings of International Conference on Learning Representations*.

[14] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[15] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

[16] Kim, S., Kim, I., Lim, S., Baek, W., Kim, C., Cho, H., Yoon, B., and Kim, T. (2019). Scalable neural architecture search for 3d medical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 220–228. Springer.

[17] Kuleshov, V., Fenner, N., and Ermon, S. (2018). Accurate uncertainties for deep learning using calibrated regression. In *International Conference on Machine Learning*, pages 2796–2804.

[18] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413.

[19] Lee, J., Lee, Y., Kim, J., Yang, E., Hwang, S. J., and Teh, Y. W. (2020). Bootstrapping neural processes. *arXiv preprint arXiv:2008.02956*.

[20] Lee, K., Lee, K., Lee, H., and Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 7167–7177.

[21] Liang, S., Li, Y., and Srikant, R. (2018). Enhancing the reliability of out-of-distribution image detection in neural networks. In *6th International Conference on Learning Representations, ICLR 2018*.

[22] Ma, X., Li, B., Wang, Y., Erfani, S. M., Wijewickrema, S., Schoenebeck, G., Houle, M. E., Song, D., and Bailey, J. (2018). Characterizing adversarial subspaces using local intrinsic dimensionality. In *International Conference on Learning Representations*.

[23] Moon, J., Kim, J., Shin, Y., and Hwang, S. (2020). Confidence-aware learning for deep neural networks. In *international conference on machine learning*.

[24] Naeini, M. P., Cooper, G. F., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, volume 2015, page 2901. NIH Public Access.

[25] Nalisnick, E. and Smyth, P. (2017). The amortized bootstrap. In *ICML Workshop on Implicit Models.*

[26] Newton, M. A. and Raftery, A. E. (1994). Approximate Bayesian inference with the weighted likelihood bootstrap. *Journal of the Royal Statistical Society: Series B (Methodological)*, 56(1):3–26.

[27] Præstgaard, J. and Wellner, J. A. (1993). Exchangeably weighted bootstraps of the general empirical process. *The Annals of Probability*, pages 2053–2086.

[28] Reed, S., Lee, H., Anguelov, D., Szegedy, C., Erhan, D., and Rabinovich, A. (2014). Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*.

[29] Shao, J. and Tu, D. (1996). *The jackknife and bootstrap*. Springer Science & Business Media.

[30] Shin, M., Lee, Y., and Liu, J. S. (2020). Scalable uncertainty quantification via generative bootstrap sampler. *arXiv preprint arXiv:2006.00767*.

[31] Smith, L. and Gal, Y. (2018). Understanding Measures of Uncertainty for Adversarial Example Detection. In *UAI*.

[32] Snelson, E. and Ghahramani, Z. (2006). Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264.

[33] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NIPS*.

Checklist

(i) For all authors...

(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 1

(b) Did you describe the limitations of your work? [Yes] See Section 6

(c) Did you discuss any potential negative societal impacts of your work? [No] We could not found any negative societal effect of our work.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

(ii) If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] See Appendix

(b) Did you include complete proofs of all theoretical results? [Yes] See Appendix

(iii) If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 4

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report error bars except for OOD experiment

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 3.2

(iv) If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [No] NIH3T3 [3] is proprietary.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] There is no issue on this problem. NTH3T3 [3] is cell-line data.

(v) If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] Unnecessary

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Unnecessary

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] Unnecessary