Improving Deep Forest by Confidence Screening

Ming Pang[1,2], Kai-Ming Ting[3], Peng Zhao[1,2], Zhi-Hua Zhou[1,2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
[2]Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210023, China
[3]School of Engineering and Information Technology, Federation University, Australia
Email: [1,2]{pangm, zhaop, zhouzh}@lamda.nju.edu.cn [3]kaiming.ting@federation.edu.au

***Abstract*—Most studies about deep learning are based on neural network models, where many layers of parameterized nonlinear differentiable modules are trained by backpropagation. Recently, it has been shown that deep learning can also be realized by non-differentiable modules without backpropagation training called deep forest. The developed representation learning process is based on a cascade of cascades of decision tree forests, where the high memory requirement and the high time cost inhibit the training of large models. In this paper, we propose a simple yet effective approach to improve the efficiency of deep forest. The key idea is to pass the instances with high confidence directly to the final stage rather than passing through all the levels. We also provide a theoretical analysis suggesting a means to vary the model complexity from low to high as the level increases in the cascade, which further reduces the memory requirement and time cost. Our experiments show that the proposed approach achieves highly competitive predictive performance with significantly reduced time cost and memory requirement by up to one order of magnitude.**

I. Introduction

Deep learning has achieved great success in various applications, particularly with visual and speech information [1], [2]. Most studies about deep learning are based on neural network models, or more accurately, many layers of parameterized nonlinear differentiable modules that can be trained by backpropagation. By recognizing that the keys of deep learning may lie in the layer-by-layer processing, in-model feature transformation and sufficient model complexity, Zhou and Feng [3] proposed a deep learning method named gcForest, which is realized by non-differentiable modules without backpropagation training.

Essentially, gcForest is a novel decision tree ensemble method with predictive accuracy highly competitive to deep neural networks in a broad range of tasks. Besides, gcForest is much easier to train because it has fewer hyper-parameters. It has been shown that gcForest can achieve high predictive accuracy on datasets across different domains by using almost the same settings of hyper-parameters. Another advantage is that the model complexity of gcForest can be determined automatically for different training datasets, enabling gcForest to perform well even on small-scale datasets. In contrast, deep neural networks usually require a huge amount of training data which prevent them from being applied to small-scale datasets.

A deep forest ensemble with a cascade structure enables gcForest to do representation learning. In this cascade structure, each level consists of an ensemble of decision tree forests [4], [5], i.e., an ensemble of ensembles [6]. Each level receives a new set of features as input which is the output of its preceding level. For textual or structural data, gcForest further enhances its representational learning ability by a technique called multi-grained scanning.

Although the results in [7] suggest that larger models might tend to offer better accuracy, Zhou and Feng [3] have not tried larger models with more grains and larger number of forests and trees (in each forest) which is limited by the high memory requirement and time cost.

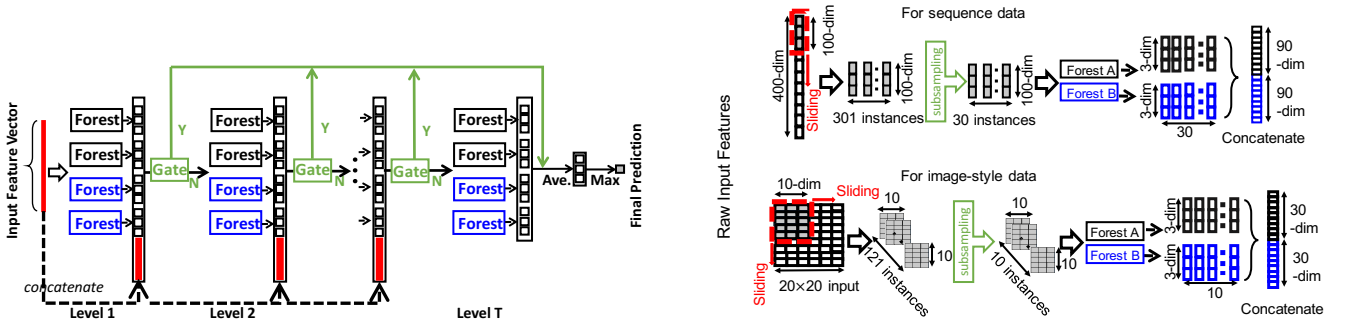
We identify that the main cause of this limitation owes much to two aspects. Firstly, gcForest passes all instances through all levels of the cascade, leading to a linear increase of time complexity w.r.t. the number of levels. Secondly, multi-grained scanning usually converts one (original) instance into hundreds or even thousands of new instances which significantly increases the number of training instances and also produces a high-dimensional input for the following cascade procedure.

To address these issues, we introduce a *confidence screening* mechanism in the general framework of deep forest, with the aim to reduce memory requirement and time cost. Specifically, the confidence screening categorizes instances at every level of the cascade into two subsets: one is easy to predict; and the other is hard. If an instance is easy to predict, its final prediction is produced at the current level; only if an instance is hard to predict, it needs to go through the next level (and potentially all levels in the cascade).

In addition, we provide a theoretical analysis which suggests a means to vary the model complexity from low to high as the level increases in the cascade. This design further reduces the memory requirement and time cost at the first few levels. Furthermore, we adopt a subsampling regime to significantly reduce the number of instances generated in the multi-grained scanning process.

In a nutshell, we propose an improved deep forest called *gcForest$_{CS}$* which is based on the confidence screening mechanism, coupled with a method to vary model complexity and the subsampling multi-grained scanning. Our experiments show that gcForest$_{\mathrm{CS}}$ achieves predictive accuracy comparable to or better than gcForest. This is achieved with up to an order of magnitude lower memory requirement and faster runtime.

The rest of this paper is organized as follows. Section II proposes our method gcForest$_{\mathrm{CS}}$. Section III gives an analysis for the confidence screening mechanism. Section IV provides the discussion. Section V reports the empirical results. Section VI studies the influence of confidence screening. Section VII concludes this paper.



(a) Cascade forest with confidence screening

(b) Feature re-representation using sliding window scanning with *subsampling*

Fig. 1: gcForest$_{\text{cs}}$: illustration of the cascade forest structure with *confidence screening* and feature re-representation using sliding window scanning with *subsampling regime*. (a) Suppose there are two random forests (black) and two completely-random forests (blue) at each level of the cascade. For a three-class problem, each forest outputs a three-dimensional class vector, which is concatenated for re-representation of the original input. Instances with high prediction confidence (Y) at level i are predicted directly—they do not go through all the levels. Only instances with low prediction confidence (N) traverse to the next level. (b) Suppose there are three classes. Raw features are 400-dim, sliding window is 100-dim and subsampling size is 30 for the sequence data; raw features are 20×20-dim, sliding window is 10×10-dim and subsampling size is 10 for the image-style data.

II. The Proposed Approach gcForest$_{\text{cs}}$

The new deep forest approach gcForest$_{\text{cs}}$ has the key *confidence screening* mechanism coupled with variable model complexity and subsampling multi-grained scanning to reduce the memory requirement and time cost of deep forest.

Rather than requiring all instances to go through all levels of cascade, we reduce the computation by requiring selective instances to go through the next level. The selection criterion is based on the prediction *confidence* which is the maximum value of the estimated class vector of one instance. For example, in a 3-class classification problem, as shown in Figure 2, the estimated class vector of an instance is $[0.7, 0.2, 0.1]$, then its prediction confidence is 0.7.

The basic idea is that an instance is pushed to the next level only if it is determined to require a higher level of learning; otherwise, it is predicted using the model at the current level. Based on this idea, we propose the deep forest structure with gates for confidence screening as shown in Figure 1(a).

A deep forest model with confidence screening can be formalized as follows. Consider the supervised learning problem of learning a mapping from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$, where $\mathcal{Y} = \{1, 2, \ldots, C\}$. Let $\mathcal{Z} = [0, 1]^C$ and training set $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ be drawn i.i.d. from

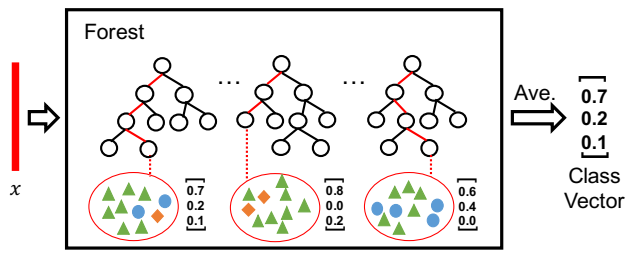


Fig. 2: Illustration of class vector generation from [3]. Each component of the final class vector is an average of the outputs of individual trees. Different shapes in leaf nodes denote different classes.

distribution $\mathcal{D}$. A deep forest model with confidence screening can be defined by a triplet $(\mathbf{h}, \mathbf{f}, \boldsymbol{\kappa})$ where

- $\mathbf{h} = (h_1, \ldots, h_T)$, where h_t is the ensemble of forests at level t, and h_t is a member of hypothesis class H_t.
- $\mathbf{f} = (f_1, \ldots, f_T)$, where f_t is the cascade of ensembles of forests up to level t.
- $\boldsymbol{\kappa} = (\kappa_1, \ldots, \kappa_T)$, where κ_t is a screener function: $\kappa_t(\mathbf{x}) = 1$ if $\mathbf{x}$ is predicted by f_t, $\kappa_t(\mathbf{x}) = 0$ otherwise.

At level $t \in \{1, \ldots, T\}$, f_t: $\mathcal{X} \to \mathcal{Z}$ is defined as follows:

$$f_t(\mathbf{x}) = \begin{cases} h_1(\mathbf{x}) & t = 1, \\ h_t\big([\mathbf{x}, f_{t-1}(\mathbf{x})]\big) & t > 1. \end{cases} \tag{1}$$

At every level t, $h_t(\cdot)$ and $f_t(\cdot)$ output a class vector $[p_1^t, \ldots, p_C^t]$, where p_i is the prediction confidence of class i. The input of h_t is $[\mathbf{x}, f_{t-1}(\mathbf{x})]$ except at level $t = 1$, where its input is $\mathbf{x}$.

Screener $\kappa_t(\cdot)$ at level t is defined based on prediction confidence and confidence threshold η_t.

$$\kappa_t(\mathbf{x}) = \begin{cases} 1 & \max_{c \in \{1,\ldots,C\}} [f_t(\mathbf{x})]_c > \eta_t, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Let $\kappa_t^{-1}(1)$ denote the set of instances such that $\kappa_t(\mathbf{x}) = 1$, which is the set of instances with high confidence at level t. $\kappa_t^{-1}(0)$ is similarly defined for those with low confidence.

At level t, if the prediction confidence of one instance is larger than threshold η_t, then its final prediction is produced at the current level; otherwise it needs to go through the next level (and potentially all levels in the cascade).

Each triplet $(\mathbf{h}, \mathbf{f}, \boldsymbol{\kappa})$ defines a deep forest model with confidence screening $g : \mathcal{X} \to \mathcal{Y}$ as follows:

$$g(\mathbf{x}) = \arg\max_{c \in \{1,\ldots,C\}} \big[f_{t'}(\mathbf{x})\big]_c, \tag{3}$$

where $t' = \arg_{t \in \{1,\ldots,T\}} \kappa_t(\mathbf{x}) = 1$.

The key issue here is how to set the confidence threshold at each level. In principle, one can define an optimization

Algorithm 1 gcForest$_{\text{cs}}$

Input: Training set S, validation set S_v, learning algorithm $\mathcal{A}$ and the maximal number of cascade levels T.
Output: The gcForest$_{\text{cs}}$ model g.
Initialize: $S_1 = S$, $\ell_0 = 1$ and $t = 1$
Process:

```
while t ≤ T do
    h_t = A(S_t).
    Get f_t according to Eq. (1).
    Get κ_t according to Eq. (2).
    Get g_t according to Eq. (3).
    Compute the validation error ℓ_t = L_{S_v}(g_t)
    if ℓ_t > ℓ_{t-1} then
        return g.
    end if
    g = g_t.
    S_{t+1} = S_t \ κ_t^{-1}(1).
    t = t + 1.
end while
return g.
```

framework in which the confidence threshold at each level is set to trade off between minimizing the expected number of instances to be passed to the next level and maximizing the expected number of instances that can be corrected at the next level (which are misclassified at the current level.) Unfortunately, finding this optimum is a difficult problem.

Instead, we use a simple rule to produce an effective cascade forest which is highly efficient. The prediction confidence threshold η_t at level t is determined automatically based on the cross-validated error rate ϵ_t of all the training instances. Let hyper-parameter $a < 1$ be a fraction of ϵ_t. All the training instances are sorted in descending order of their prediction confidences, where c_i is the prediction confidence of $\mathbf{x}_i$. η_t is set according to the following formulation:

$$\eta_t = \min\{c_k | L(\mathbf{x}_1, \ldots, \mathbf{x}_k) < a\epsilon_t,\ k \in [1, m]\}, \tag{4}$$

where $L(\mathbf{x}_1, \ldots, \mathbf{x}_k) = \frac{1}{k}\sum_{i=1}^{k} \mathbb{1}[g_t(\mathbf{x}_i) \neq y_i]$ is the error rate of the k instances with the largest prediction confidences. Note that a is the only additional hyper-parameter for confidence screening, compared with gcForest.

Variable model complexity. Because the remaining instances become increasingly hard-to-predict as the level increases, gcForest$_{\text{cs}}$ uses increasingly complex forests at high levels, i.e., gcForest$_{\text{cs}}$ increases the number of trees in each forest linearly as the number of training instances decreases.

The above strategy follows the result of the theoretical analysis in Section III. It suggests that varying the model complexity from low to high as the level increases in the cascade can lead to better generalization performance. This design further reduces memory requirement and time cost; and the model complexity only increases as the level increases when it is most needed to produce an accurate model for hard-to-predict instances.

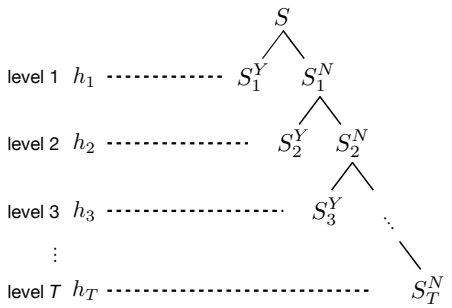


Fig. 3: The structure of the cascade forest with confidence screening. There is a total of T levels, where the instances are split into two parts at each level: the left part contains instances with high confidence, and the right part contains those with low confidence and thus needs to be further processed at higher levels.

Subsampling multi-grained scanning. Multi-grained scanning in gcForest [3] increases the memory consumption and runtime heavily. To address this issue, as suggested in [3], we use *subsampling* in multi-grained scanning. Specifically, we randomly sample from the set of the converted instances produced in multi-grained scanning. As Figure 1(b) illustrates, subsampling not only reduces the number of converted instances, but also reduces the number of dimensions of the transformed features by an order of magnitude from 903 to 90 dimensions. At each level, subsampling multi-grained scanning generates new transformed features, and the features from the recent three levels are concatenated to classify the remaining instances which are fewer and "harder".

Algorithm 1 summarizes the proposed gcForest$_{\text{cs}}$.[1]

III. Analysis

In this section, we provide an analysis for both the confidence screening and variable model complexity. In particular, we are interested in the effect of splitting all instances into two parts according to the prediction confidences, and thus, we ignore the effect of concatenation of previous label predictions in the analysis.

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ be the training set of size m drawn according to the underlying distribution $\mathcal{D}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{Y} = \{1, 2, \ldots C\}$ is the associated class label. Let the loss function be $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$. For a given hypothesis h, its risk $R(h)$ and empirical risk $\hat{R}_S(h)$ are:

$$R(h) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(h(\mathbf{x}), y)], \quad \hat{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i).$$

An illustration of the cascade structure is given in Figure 3. Suppose the cascade has T levels, and has $\mathbf{h} = (h_1, \ldots, h_T)$ with the classifier at level t $h_t : \mathcal{X} \to [-1, +1]$; and h_t is a member of the hypothesis set $\mathcal{H}_t$. Note that we consider the two-class problem only here for ease of analysis. Furthermore, we denote $\mathscr{H}_t = \{\mathbf{x} \to \kappa_t(\mathbf{x})h_t(\mathbf{x}) : \kappa_t \in \mathcal{K}_t, h_t \in \mathcal{H}_t\}$ as the family of the products of the screener function and

[1] Note that subsampling multi-grained scanning can be used to provide a new representation at each level.

the classifier at level t. Then we have the following results regarding the generalization bound of the learned model.

Theorem 1. Assume that the function in $\mathcal{H}_t$ takes values in $[-1,+1]$ for all $t \in \{1,\ldots,T\}$, and the training sample S is of size m drawn i.i.d. from underlying distribution $\mathcal{D}$. Then, for any $\delta > 0$, with probability at least $1-\delta$, the following holds for all h with T levels cascade forest:

$$R(h) \leq \hat{R}_S(h) + \sum_{t=1}^{T} \min\left(4\hat{\mathfrak{R}}_S(\mathscr{H}_t), \frac{m_t}{m}\right) + \tilde{O}\left(T\sqrt{\frac{\log T}{m}}\right),$$

where $\hat{\mathfrak{R}}_S$ is the empirical Rademacher complexity, and $m_t = |S_t^Y|$ is the number of screened instances at level t.

Proof. First, we introduce the convex ensembles with multiple hypothesis set $g_{\boldsymbol{\alpha}}$. For any $\boldsymbol{\alpha} \in \Delta_T$, denote $g_{\boldsymbol{\alpha}}$ as follows,

$$g_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \kappa_t(\mathbf{x}) h_t(\mathbf{x}),$$

where Δ_T is the simplex in $\mathbb{R}^T$. Fix $\rho > 0$, since $g_{\boldsymbol{\alpha}}$ is a convex combination of the mappings $\mathbf{x} \to \kappa_t(\mathbf{x})h_t(\mathbf{x})$, and note that $h(\mathbf{x}) = \sum_{t=1}^{T} \kappa_t(\mathbf{x})h_t(\mathbf{x})$, using Theorem 1 in [8], we can obtain

$$\begin{aligned} R(h) \leq & \inf_{\boldsymbol{\alpha}\in\Delta_T} \left[\hat{R}_{S,\rho}(g_{\boldsymbol{\alpha}}) + \frac{4}{\rho}\sum_{t=1}^{T} \alpha_t \hat{\mathfrak{R}}_S(\mathscr{H}_t)\right] \\ & + C(m,p) + \sqrt{\frac{\log(4/\delta)}{2m}}, \end{aligned} \tag{5}$$

where $C(m,\rho) = \frac{2}{\rho}\sqrt{\frac{\log T}{m}} + \sqrt{\frac{\log T}{\rho^2 m}\log\left(\frac{\rho^2 m}{\log T}\right)}$, and $\hat{R}_{S,\rho}(g_{\boldsymbol{\alpha}}) = \frac{1}{m}\sum_{t=1}^{T}\sum_{\kappa(\mathbf{x}_i,k)=1} \mathbb{I}[y_i \alpha_t h_t(\mathbf{x}_i) < \rho]$. $\mathbb{I}[e(\cdot)]$ is the indicator function taking 1 if $e(\cdot)$ is true; and 0 otherwise.

The second step is to provide the upper bound for the first term in r.h.s. of Eq. (5). Following the analysis in [9],

$$\begin{aligned} & \inf_{\boldsymbol{\alpha}\in\Delta_T} \left[\hat{R}_{S,\rho}(g_{\boldsymbol{\alpha}}) + \frac{4}{\rho}\sum_{t=1}^{T} \alpha_t \hat{\mathfrak{R}}_S(\mathscr{H}_t)\right] \\ \leq & \hat{R}_S(h) + \sum_{t=1}^{T} \min\left(4\hat{\mathfrak{R}}_S(\mathscr{H}_t), \frac{m_t}{m}\right) \\ & + \min_{\substack{\mathcal{I}\subseteq\mathcal{T},\\ |\mathcal{I}|\geq|\mathcal{T}|-1/\rho}} \sum_{t\in\mathcal{I}} \left(\frac{m_t}{m} - 4\hat{\mathfrak{R}}_S(\mathscr{H}_t)\right) \end{aligned} \tag{6}$$

Hence, we complete the proof by combining (5) and (6),

$$\begin{aligned} R(h) \leq & \hat{R}_S(h) + \sum_{t=1}^{T} \min\left(4\hat{\mathfrak{R}}_S(\mathscr{H}_t), \frac{m_t}{m}\right) + C(m,\rho) \\ & + \min_{\substack{\mathcal{I}\subseteq\mathcal{T},\\ |\mathcal{I}|\geq|\mathcal{T}|-1/\rho}} \sum_{k\in\mathcal{I}} \left(\frac{m_t}{m} - 4\hat{\mathfrak{R}}_S(\mathscr{H}_t)\right) + \sqrt{\frac{\log(4/\delta)}{2m}}, \end{aligned}$$

where $C(m,\rho) = \frac{2}{\rho}\sqrt{\frac{\log T}{m}} + \sqrt{\frac{\log T}{\rho^2 m}\log\left(\frac{\rho^2 m}{\log T}\right)}$; and $\mathcal{T} = \{k : m_t/m > 4\hat{\mathfrak{R}}_S(\mathscr{H}_t)\}$, the set of level t whose screening ratio is greater than $4\hat{\mathfrak{R}}_S(\mathscr{H}_t)$.

To simplify the presentation, we ignore the non-leading terms and only keep the terms regarding cascade level T, instance number m and Rademacher complexity terms, and obtain the simpler form as specified in Theorem 1. □

Remark. The generalization error bound in Theorem 1 sheds light on the cascade structure design. Note that the second term is the minimization of screening ratio m_t/m and complexity term $4\hat{\mathfrak{R}}_S(\mathscr{H}_t)$. Because most of the instances will be screened in the first few levels, the screening ratio is large. Hence, one should reduce the corresponding complexity term in these few levels in order to make the generalization error bound tighter. This is the theoretical basis in which we vary the model complexity at a level in the cascade from low to high (by increasing the ensemble size) as the level t increases.

IV. Discussion

The high memory requirement and time cost of gcForest can be tackled by exploiting distributed implementation [10] or hardware facilitation. We believe, however, there is demand to tackle them via algorithmic improvement.

The cascade procedure with confidence screening has some connection with two lines of research. First, confidence screening is related to boosted cascade [11] which aims to reject many negative instances and has achieved success in visual object detection problems. Although the cascade structure appears to be similar on the surface, the boosted cascade procedure is not suitable for classification tasks because the nature of the object detection tasks is different.

Second, there are some studies which add the output of one classifier as an additional input for another classifier in a series of multiple classifiers to improve the accuracy of a single classifier [12], [13]. Like gcForest, these methods pass all the instances through all the classifiers which is inefficient.

The subsampling strategy is related to sampling strategies for bag-of-features image classification [14], [15]. By treating an image as a collection of independent patches, random sampling gives equal or better representative selection than the sophisticated multiscale interest operators. In this paper, we use a simple random sampling strategy and show that it works in the context of deep forest. It is interesting to explore other sampling strategies [16], [17].

In addition, using more features for the low-confidence instances is related to the time-efficient feature extraction approach [18]. For test instances, this approach only extracts cheap and sufficient features, and when the classification confidence is high enough, the test instance will be classified. There are two main differences between this work and our approach. First, their goal is to reduce test time cost, while our goal is to reduce both train and test time cost of gcForest. Second, features are given with known feature extraction time costs in their setting, while the transformed features in deep forest are unknown before multi-grained scanning.

Recently, Utkin and Ryabinin [19] modified gcForest and proposed a Siamese deep forest as an alternative to the Siamese neural networks to solve the metric learning tasks. Because our target is to improve the efficiency of deep forest, our method can help to improve the efficiency of Siamese deep forest and other modified applications of gcForest as well.

V. EXPERIMENTS

The goal is to validate that gcForest$_{cs}$ can achieve predictive accuracy comparable to or better than gcForest with much less space and time cost.

Parameter Settings. In all experiments, both gcForest and gcForest$_{cs}$ use the *same* cascade structure. Every level consists of v random forests and v completely-random forests [5] in the experiments, where $v = 4$ with multi-grained scanning and $v = 1$ without. The class vector of each forest is generated by three-fold cross validation.

For gcForest, every forest has 500 trees as recommended in [3]. For gcForest$_{cs}$, each forest at the first level has w trees and the number of trees increases linearly as the number of instances decreases at the subsequent levels. In the experiments without multi-grained scanning, $w = 20,\ 50,\ 100$ to examine their effects; and $w = 100$ in the experiments with multi-grained scanning.

The number of cascade levels stops increasing when the current level does not improve the accuracy of the previous level for both gcForest$_{cs}$ and gcForest.

For gcForest$_{cs}$, the prediction confidence threshold η at each level is determined automatically according to Equation 4. Hyper-parameter a is set according to a simple rule as follows. If subsampling multi-grained scanning is used, $a = 1/20$. Otherwise, a is set according to the training accuracy of the first level ϵ_1. If $\epsilon_1 > 90\%$, $a = 1/10$; otherwise, $a = 1/3$.

In (subsampling) multi-grained scanning, gcForest uses three window sizes with sizes of $\lfloor d/16 \rfloor$, $\lfloor d/8 \rfloor$, $\lfloor d/4 \rfloor$; gcForest$_{cs}$ uses one window size with size $\lfloor d/16 \rfloor$ for d raw features. Note that gcForest$_{cs}$ could adopt multiple window sizes which might produce a better accuracy as suggested by Zhou and Feng [3]. Nevertheless, it is sufficient to use only one window size for gcForest$_{cs}$ to achieve a satisfactory accuracy.

Datasets. Experiments are performed on *all* the datasets used by gcForest (except three small datasets which have little to do with our purpose of improving the efficiency), i.e., LETTER, ADULT, IMDB, MNIST, sEMG, CIFAR-10.

Evaluation metrics. We adopt the predictive accuracy as the classification performance measurement which is suitable for these balanced datasets. Training time, test time and memory usage are used to evaluate the efficiency.

Hardware. In the experiments without multi-grained scanning, we use a machine with 4×2.10 GHz CPUs and 32GB main memory. In the experiments with multi-grained scanning, we use a machine with 28×2.40 GHz CPUs and 756GB main memory. This is because 32GB main memory is not enough for the multi-grained scanning procedure of gcForest (although gcForest$_{cs}$ has no barriers).

The experiments are divided into two categories: with and without multi-grained scanning, described in the following two subsections.

A. Results with Multi-Grained Scanning

The datasets sEMG, MNIST and CIFAR10 are used here because they hold spatial or sequential relationships among the raw features; and the other datasets do not.

TABLE I: Comparison results (with multi-grained scanning) between gcForest$_{cs}$ and gcForest on accuracy, training time, test time (in CPU seconds) and memory usage (in megabytes), with given test sets.

Datasets	Method	Accuracy	Training time	Test time	Memory
sEMG	gcForest$_{cs}$	72.59	1547.62	77.48	4348
	gcForest	71.30	34323.78	2288.29	41789
MNIST	gcForest$_{cs}$	99.26	1060.65	9.64	4997
	gcForest	99.26	27840.39	464.27	50518
CIFAR10	gcForest$_{cs}$	62.62	13341.68	667.08	6875
	gcForest	61.78	63068.32	2102.71	73826

TABLE II: Comparison results (without multi-grained scanning) between gcForest$_{cs}$ and gcForest on accuracy (%), training time, test time (in CPU seconds) and memory usage (in megabytes). 10 test runs were conducted and the average accuracies are presented.

Datasets	Method	Accuracy	Training time	Test time	Memory
LETTER	gcForest$_{cs}$(20)	96.42	13.39	0.41	206
	gcForest$_{cs}$(50)	96.93	17.16	1.25	472
	gcForest$_{cs}$(100)	97.08	75.23	2.23	915
	gcForest	97.08	86.42	3.17	4526
ADULT	gcForest$_{cs}$(20)	86.04	27.47	2.18	173
	gcForest$_{cs}$(50)	86.04	45.19	4.12	351
	gcForest$_{cs}$(100)	86.11	95.48	6.86	648
	gcForest	86.06	198.85	12.24	3002
IMDB	gcForest$_{cs}$(20)	89.19	590.79	16.63	1518
	gcForest$_{cs}$(50)	89.40	974.07	23.19	1802
	gcForest$_{cs}$(100)	89.57	1623.11	32.05	1992
	gcForest	89.20	11633.65	152.20	3750

Table I shows that gcForest$_{cs}$ achieves comparable or even better predictive accuracy than gcForest with about an order of magnitude less memory and faster runtime. The only exception is the CIFAR-10 dataset, where the training and testing times are still 4 and 3 times faster, respectively.

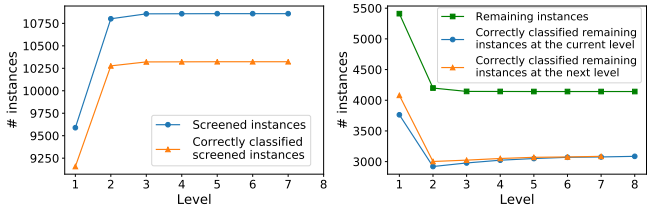
Interestingly, if gcForest adopts subsampling multi-grained scanning at each level (the same as gcForest$_{cs}$) instead of multi-grained scanning, its predictive accuracy will degrade heavily, e.g., it achieves 67.78% on sEMG which is much lower than the original gcForest and gcForest$_{cs}$. Thus we report the results of the original gcForest only. This outcome further verifies the effectiveness of confidence screening.

B. Results without Multi-Grained Scanning

Here we conduct experiments without multi-grained scanning on the datasets that do not hold spatial or sequential relationships among the raw features. gcForest$_{cs}$ uses different numbers of trees of each forest at the first level, i.e., 20, 50 and 100 to examine their rates of improvements.

As shown in Table II, gcForest$_{cs}$ improves its accuracy as window size increases; but the maximal gap among them is less than 0.7%. In contrast, gcForest has a large accuracy gap, e.g., gcForest(20) achieves 88.21% on IMDB which is much lower than 89.20% achieved by gcForest(500). Because the accuracy of gcForest with fewer trees is not competitive, we report its accuracy in which each forest has 500 trees only.

Table II shows that gcForest$_{cs}$(100) achieves accuracies comparable to or better than gcForest on all three datasets.



(a) Screened instances (b) Remaining instances

Fig. 4: IMDB: Screened instances and remaining instances at each level of gcForest$_{cs}$. (a) The accumulated numbers of screened instances and correctly classified screened instances up to a certain level of gcForest$_{cs}$; (b) The number of remaining instances at each level of gcForest$_{cs}$. At each level, the remaining instances are passed to the next level. The number of correctly classified remaining instances increases from the current level to the next level.

VI. Influence of Confidence Screening

In this section, we aim to study the influence of confidence screening that links the decreasing number of instances with the improvement at each level in terms of accuracy, memory and runtime. The examples are based on IMDB, and the results are similar on other datasets.

As shown in Figure 4(a), gcForest$_{cs}$ screens 64% of the test instances of IMDB at the first level. At the last level, only 28% test instances are passed through all the levels. In contrast, all the test instances are passed through all the levels in gcForest. The results are similar on training. This leads directly to the high reduction in memory and runtime.

We further demonstrate how confidence screening influences the classification result. As Figure 4(a) shows, the accumulated number of screened instances increases as the number of levels increases. Because instances are screened if their predictions have high confidence, the accuracy of the screened instances is about 95% which is much higher than the overall accuracy ($< 90\%$). Interestingly, gcForest$_{cs}$ and gcForest have the same predictions on the screened instances except two of them.

The result of the remaining instances is shown in Figure 4(b): the blue line plots the number of remaining instances which are correctly classified at the current level; and the orange line represents the number of those correctly classified at the next level. Figure 4(b) shows that the accuracy of remaining instances increases when they are passed from the current level to the next level—due to the model used in the next level. At the final level, the number of correctly classified instances of gcForest$_{cs}$ is 80 instances more than that of gcForest. This outcome shows that confidence screening encourages models at each level to better focus on the hard-to-predict instances that leads directly to their better predictions.

VII. Conclusions

We propose a more efficient Deep Forest approach called gcForest$_{cs}$ which has significantly smaller memory requirement and runs faster than gcForest [3]. We first identify two deficiencies of gcForest and then introduce a confidence screening mechanism to address these issues. The confidence screening splits the instances into easy-to-predict and hard-to-predict subsets at each level of cascade, which reduces the number of instances passed to the next level. This significantly reduces the training and testing times of forests at each level. Our theoretical analysis suggests to vary the model complexity from low to high as the level increases in the cascade. This design further reduces memory requirement and time cost at the first few levels; the model complexity increases as the level increases to cope with hard-to-predict instances.

The effectiveness of the approach is validated in our evaluation that it achieves more with less, i.e., gcForest$_{cs}$ has predictive accuracy comparable to or better than gcForest, with up to one order of magnitude smaller time and memory costs.

Acknowledgment: This research was supported by NSFC (61751306) and the 111 Program (B14020).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] Z.-H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," in *IJCAI*, 2017, pp. 3553–3559.

[4] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[5] F. T. Liu, K. M. Ting, Y. Yu, and Z.-H. Zhou, "Spectrum of variable-random trees," *Journal of Artificial Intelligence Research*, vol. 32, pp. 355–384, 2008.

[6] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL: CRC, 2012.

[7] Z.-H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," *CoRR*, vol. abs/1702.08835, 2017.

[8] C. Cortes, M. Mohri, and U. Syed, "Deep boosting," in *ICML*, 2014, pp. 1179–1187.

[9] G. DeSalvo, M. Mohri, and U. Syed, "Learning with deep cascades," in *ALT*, 2015, pp. 254–269.

[10] Y.-L. Zhang, J. Zhou, W. Zheng, J. Feng, L. Li, Z. Liu, M. Li, Z. Zhang, C. Chen, X. Li, and Z.-H. Zhou, "Distributed deep forest and its application to automatic detection of cash-out fraud," *CoRR*, vol. abs/1805.04234, 2018.

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *CVPR*, 2001, pp. 511–518.

[12] J. Gama and P. Brazdil, "Cascade generalization," *Machine Learning*, vol. 41, no. 3, pp. 315–343, 2000.

[13] H. Zhao and S. Ram, "Constrained cascade generalization of decision trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 6, pp. 727–739, 2004.

[14] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *ECCV*, 2006, pp. 490–503.

[15] F. Shi, E. Petriu, and R. Laganiere, "Sampling strategies for real-time action recognition," in *CVPR*, 2013, pp. 2595–2602.

[16] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *ICML*, 2009, pp. 1113–1120.

[17] A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan, "The big data bootstrap," in *ICML*, 2012.

[18] L.-P. Liu, Y. Yu, Y. Jiang, and Z.-H. Zhou, "TEFE: a time-efficient approach to feature extraction," in *ICDM*, 2008, pp. 423–432.

[19] L. V. Utkin and M. A. Ryabinin, "A siamese deep forest," *Knowledge-Based Systems*, vol. 139, pp. 13–22, 2018.