Deep Forest

Zhi-Hua Zhou, Ji Feng
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
{zhouzh, fengj}@lamda.nju.edu.cn

Abstract

Current deep learning models are mostly built upon neural networks, i.e., multiple layers of parameterized differentiable nonlinear modules that can be trained by backpropagation. In this paper, we explore the possibility of building deep models based on non-differentiable modules such as decision trees. After a discussion about the mystery behind deep neural networks, particulary by contrasting it with shallow neural networks and traditional machine learning techniques such as decision trees and Boosting machines, we conjecture that the success of deep neural networks owes much to three characteristics, i.e., layer-by-layer processing, in-model feature transformation and sufficient model complexity. On one hand, our conjecture may offer inspirations for theoretical understandings of deep learning; on the other hand, to verify the conjecture, we propose an approach which generates deep forest holding these characteristics. This is a decision tree ensemble approach, with less hyper-parameters than deep neural networks, and its model complexity can be automatically determined in a data-dependent way. Experiments show that its performance is quite robust to hyper-parameter settings, such that in most cases, even across different data from different domains, it is able to achieve excellent performance by using the same default setting. This study opens the door of deep learning based on non-differentiable modules without gradient-based adjustment, and exhibits the possibility of constructing deep models without backpropagation.

Keywords: *Deep Forest, Deep Learning, Machine Learning, Ensemble Methods, Decision Trees*

1 Introduction

Deep learning [12] has become a hotwave in various domains. While, what is deep learning? Answers from the crowd are very likely to be that "*deep learning is a subfield of machine learning that uses deep neural networks*" (quoted from [40]). Actually, the great success of deep neural networks (DNNs) in tasks involving visual and audio information led to the rise of deep learning, and almost all current deep learning applications are built upon neural network models, or more technically, multiple layers of parameterized differentiable nonlinear modules that can be trained by backpropagation.

Though deep neural networks are powerful, they have many deficiencies. First, DNNs are with too many hyper-parameters, and the learning performance depends seriously on careful parameter tuning. Indeed, even when several authors all use convolutional neural networks [22,25,39], they are actually using different learning models due to the many different options such as the convolutional layer structures. This fact makes the training of DNNs very tricky, and theoretical analysis of DNNs extremely difficult because of too many interfering factors with almost infinite configurational combinations. Second, it is well known that the training of DNNs requires a huge amount of training data, and thus, DNNs can hardly be applied to tasks where only small-scale training data are available. Note that even in the big data era, many real tasks are still with insufficient amount of training data due to the high cost of labeling, leading to inferior performance of DNNs in these tasks. Third, the network architecture has to be determined before training, and thus, the model complexity is determined in advance. Actually, deep models are usually overly complicated than what are really needed, as verified by the observation that recently there are many reports about DNNs performance improvement by adding shortcut connection [14,41], pruning [13, 30], binarization [7, 35], etc., as these operations simplify the original networks and actually decrease model complexity. It might be better if the model complexity can be determined automatically in a data-dependent way. Furthermore, it is well known that neural networks are black-box models whose decision processes are hard to explain. It is also noteworthy that although DNNs have been well developed, there are still many tasks on which DNNs are inferior; for example, Random Forest [5] or XGBoost [6] are winners on many Kaggle competition tasks.

In order to tackle complicated learning tasks, learning models are likely have to go deep. Current deep models, however, are always built upon neural networks. As discussed above, there are good reasons to explore non-NN style deep models, or in other words, to consider whether deep learning can be realized with other modules, as they have their own advantages and may exhibit great potential if being able to go deep. In particular, considering that neural networks are multiple layers of parameterized *differentiable* nonlinear modules, whereas not all properties in the world are differentiable or best modelled as differentiable, in this paper we attempt to address the question: "*Can deep learning be realized with non-differentiable modules*?"

The answer to the question may help understand important issues such as (1) deep models ?= DNNs, or, must deep models be constructed with differentiable modules? Note that there are researches involve non-differentiable activation functions in DNNs; however, they usually use differentiable functions that upper-bound the non-differentiable ones for relaxation in the optimization/learning process, and thus, actually they still work with differentiable modules. (2) Is it possible to train deep models without backpropagation? Note that backpropagation and gradient-based adjustment require differentiability. (3) Is it possible to enable deep models win tasks on which now other models such as random forest or XGBoost are superior? Actually, the machine learning community have developed lots of learning modules, whereas many of them are non-differentiable and can hardly be tuned by gradient-based adjustment; it would be interesting to know whether it is possible to construct deep models based on these modules.

In this paper, we extend our preliminary research [52] which proposes the gcForest (multi-Grained Cascade Forest) approach for constructing deep forest, a non-NN style deep model. This is a novel decision tree ensemble, with a cascade structure which enables representation learning by forest. Its representational learning ability can be further enhanced by multi-grained scanning, potentially enabling gcForest to be contextual or structural aware. The cascade levels can be automatically determined such that the model complexity can be determined in a data-dependent way; this enables gcForest to work well even with small-scale data, and enables users to control training costs according to computational resource available. The gcForest has much less hyper-parameters than DNNs, and its performance is quite robust to hyper-parameter settings; our experiments show that in most cases, it is able to get excellent performance by using the default setting, even across different data from different domains.

Section 2 briefly introduces ensemble learning. Section 3 explains our design motivations by analyzing why deep learning works. Section 4 proposes our approach, followed by experiments reported in Section 5. Section 6 briefly presents a real application. Section 7 discusses on some related work. Section 8 raises some issues for future exploration, followed by concluding remarks in Section 9.

2 Ensemble Learning and Diversity

Ensemble learning [50] is a machine learning paradigm where multiple learners (e.g., classifiers) are trained and combined for a task. It is well known that an ensemble can usually achieve better generalization performance than single learners.

To construct a good ensemble, the individual learners should be *accurate* and *diverse*. Combining only accurate learners is often inferior to combining some accurate learners with some relatively weaker ones, because the complementarity is more important than pure accuracy. Actually, a beautiful equation has been theoretically derived from *error-ambiguity decomposition* [24]:

$$E = \bar{E} - \bar{A} \,, \tag{1}$$

where E denotes the error of an ensemble, $\bar{E}$ denotes the average error of individual classifiers in the ensemble, and $\bar{A}$ denotes the average *ambiguity*, later called *diversity*, among the individual classifiers. This offers a general guidance for ensemble construction; however, it could not be taken as an objective function for optimization, because the *ambiguity* is mathematically defined in the derivation and cannot be operated directly. Later, the ensemble community have designed many diversity measures, but none has been well-accepted as the *right* definition for diversity [50], and "*what is diversity*?" remains the holy grail problem in this area.

In practice, diversity enhancement is based on randomness injection during training. Roughly speaking, there are four major category of strategies [50]. The first is *data sample manipulation*, which works by generating different data samples for different individuals, e.g., bootstrap sampling for Bagging [2] and sequential importance sampling for AdaBoost [10]. The second is *input feature manipulation*, which works by generating different feature subspaces for different individuals, e.g., Random Subspace [17] randomly picks different subset of features for different individuals. The third is *learning parameter manipulation*, which works by using different parameter settings of the base learning algorithm to generate different individuals, e.g., different initial weights can be used for different individual neural networks. The fourth is *output representation manipulation*, which works by using different output representations for different individuals, e.g., ECOC [8] employs error-correcting output codes, whereas Flipping Output [4] randomly switches labels of training instances. Different strategies can be used together. No strategies is always effective, e.g., data sample manipulation does not work well with *stable learners* whose performance does not significantly change according to slight modification of training data. More information about ensemble learning can be found in [50]. Our gcForest can be viewed as a decision tree ensemble approach that utilizes almost all categories of strategies for diversity enhancement.

3 What's Crucial for Deep Model

It is widely recognized that the *representation learning* ability is crucial for the success of deep neural networks. What is crucial for representation learning in DNNs? We believe that the answer is *layer-by-layer processing*. Figure 1 provides an illustration simulated from a figure in [12], where features of higher levels of abstract emerge as the layer goes up from the bottom.

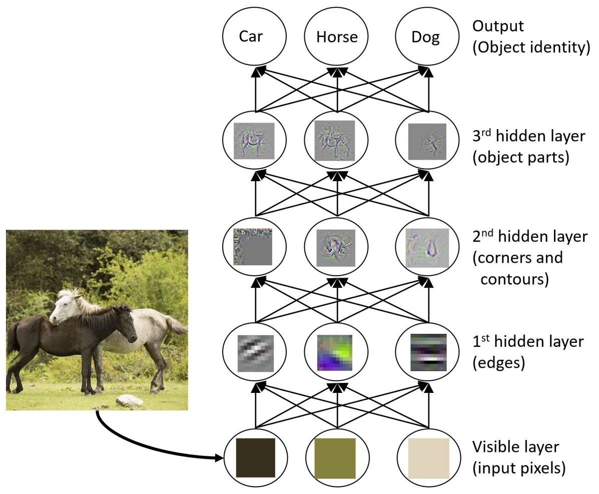

Car
Horse
Dog
Output (Object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

Figure 1. Illustration of the layer-by-layer processing in deep neural networks: Features of higher levels of abstract emerge as the layer goes up from the bottom.

Considering that if other issues fixed, large model complexity (or more accurately, model capacity) generally leads to strong learning ability, maybe it sounds reasonable to attribute the successes of DNNs to the huge model complexity. This, however, cannot explain the fact that why shallow networks are not as successful as deep ones, as one can increase the complexity of shallow networks to almost arbitrarily high by adding nearly infinite number of hidden units. Consequently, we believe that the model complexity itself cannot explain the success of DNNs. Instead, we conjecture that the layer-by-layer processing is one of the most important factors behind DNNs because shallow networks, no matter how large their complexity can be, cannot do layer-by-layer processing. This conjecture offers an important inspiration for the design of gcForest.

Then, how to explain the fact that traditional learning models that can do layer-by-layer processing, e.g., decision trees and Boosting machines, are not as successful as DNNs? We believe the distinguishing factor lies in the fact that, in contrast to DNNs where new features are generated as illustrated in Figure 1, decision trees and Boosting machines always work on the original feature representation during the learning process, or in other words, there is no *in-model* feature transformation. Moreover, in contrast to DNNs that can be endowed with arbitrarily high model complexity, decision trees and Boosting machines can only have limited model complexity. Though model complexity itself does not give rise to successes of DNNs, it is still important because large model capacity is needed if one wants to exploit big training data.

Overall, we conjecture that behind the mystery of DNNs there are three crucial characteristics, i.e., layer-by-layer processing, in-model feature transformation, and sufficient model complexity. To verify our conjecture, in next section we will try to endow these characteristics to non-NN style deep models.

4 The gcForest Approach

4.1 Cascade Forest Structure

To realize layer-by-layer processing, gcForest employs a cascade structure, as illustrated in Figure 2, where each level of cascade receives feature information processed by its preceding level.

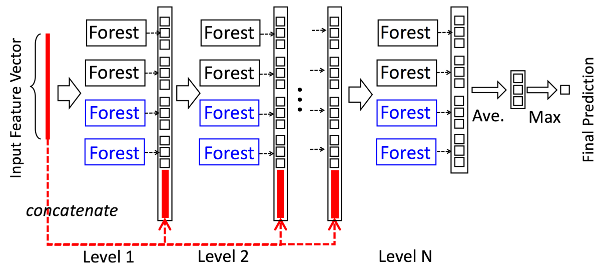


Figure 2. Illustration of the cascade forest structure. Suppose each level of the cascade consists of two random forests (black) and two completely-random forests (blue). Suppose there are three classes to predict; thus, each forest will output a three-dimensional class vector, which is then concatenated for re-representation of the input.

Each level is an ensemble of decision tree forests, i.e., an *ensemble of ensembles*. Here, we include different types of forests to encourage the *diversity*. For simplicity, suppose that we use two completely-random forests and two random forests [5]. Each completely-random forest contains 500 completely-random trees [28], generated by randomly assigning a feature for split at each node, and growing tree till pure leaf, i.e., each leaf node contains only the same class of instances. Similarly, each random forest contains 500 trees, by randomly picking $\sqrt{d}$ number of features as candidate (d is the number of input features) and selecting the one with the best *gini* value for split. The number of trees in each forest is a hyper-parameter.

Given an instance, each forest can produce an estimate of class distribution, by counting the percentage of different classes of training examples at the leaf node where the concerned instance falls, and then averaging across all trees in the same forest, as illustrated in Figure 3.

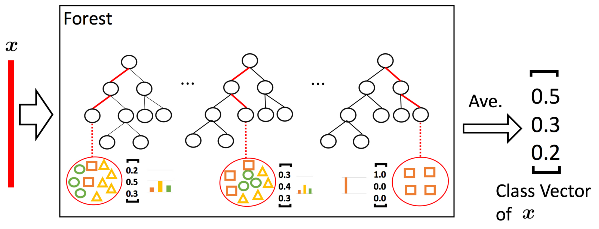


Figure 3. Illustration of class vector generation. Different marks in leaf nodes imply different classes; red color highlights paths along which the concerned instance traverses to leaf nodes.

The estimated class distribution forms a class vector, which is then concatenated with the original feature vector to input to the next level. For example, suppose there are three classes, then each of the four forests will produce a three-dimensional class vector; consequently, the next level will receive 12 ($= 3 \times 4$) augmented features.

Note that here we take the simplest form of class vectors, i.e., the class distribution at the leaf nodes into which the concerned instance falls. It is evident that such a small number of augmented features may deliver very limited augmented information, and it is very likely to be drown out when the original feature vectors are high-dimensional. We will show in experiments that such a simple feature augmentation is already beneficial, and it is expectable that more profit can be obtained if more augmented features are involved. Actually, it is apparent that more features may be incorporated, such as class distribution of the parent nodes which express prior distribution, the sibling nodes which express complementary distribution, etc. We leave these possibilities for future exploration.

To reduce the risk of overfitting, class vector produced by each forest is generated by k-fold cross validation. In detail, each instance will be used as training data for $k - 1$ times, resulting in $k - 1$ class vectors, which are then averaged to produce the final class vector as augmented features for the next level of cascade. After expanding a new level, the performance of the whole cascade can be estimated on validation set, and the training procedure will terminate if there is no significant performance gain; thus, the number of cascade levels can be automatically determined. Note that the training error rather than cross validation error can also be used to control the cascade growth when the training cost is concerned or limited computation resource available. In contrast to most deep neural networks whose model complexity is fixed, gcForest adaptively decides its model complexity by terminating training when adequate. This enables it to be applicable to different scales of training data, not limited to large-scale ones.

4.2 Multi-Grained Scanning

Deep neural networks are powerful in handling feature relationships, e.g., convolutional neural networks are effective on image data where spatial relationships among the raw pixels are critical; recurrent neural networks are effective on sequence data where sequential relationships are critical. Inspired by this recognition, we enhance cascade forest with a procedure of multi-grained scanning.

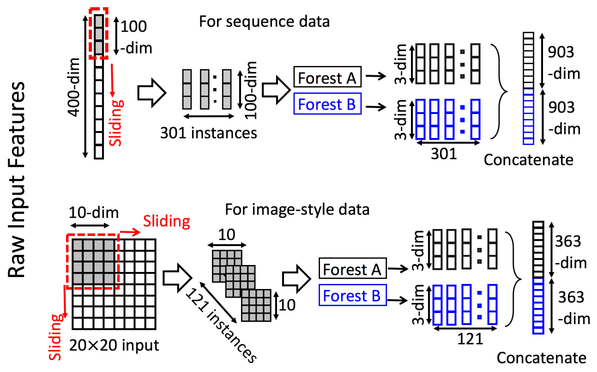


Figure 4. Illustration of feature re-representation using sliding window scanning. Suppose there are three classes, raw features are 400-dim, and sliding window is 100-dim.

As Figure 4 illustrates, sliding windows are used to scan the raw features. Suppose there are 400 raw features and a window size of 100 features is used. For sequence data, a 100-dimensional feature vector will be generated by sliding the window for one feature; in total 301 feature vectors are produced. If the raw features are with spacial relationships, such as a 20×20 panel of 400 image pixels, then a 10×10 window will produce 121 feature vectors (i.e.,

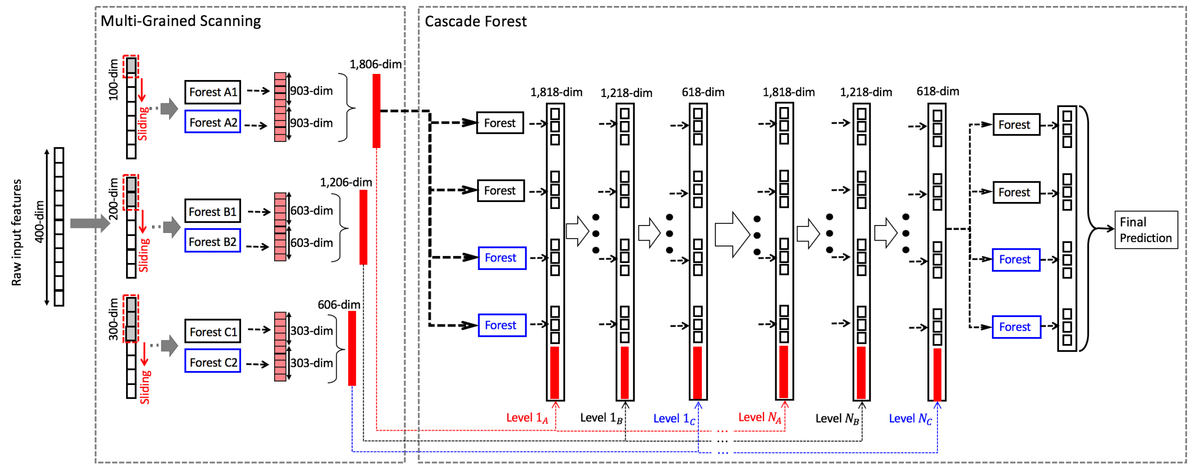


Figure 5. Overall procedure of gcForest. Suppose there are three classes, raw features are 400-dim, and three sizes of sliding windows are used.

121 10 × 10 panels). All feature vectors extracted from positive/negative training examples are regarded as positive/negative instances, which will then be used to generate class vectors like in Section 4.1: instances extracted from the same size of windows will be used to train a completely-random forest and a random forest, and then the class vectors are generated and concatenated as transformed features. As illustrated in Figure 4, for three classes, 301 three-dimensional class vectors are produced by each forest, leading to an 1,806-dimensional transformed feature vector corresponding to each 400-dimensional raw feature vector.

For instances extracted from the windows, we simply assign them with the label of the original training example. Some label assignments are inherently incorrect. For example, suppose the original training example is a positive image about "car"; it is clearly that many extracted instances do not contain a car, and therefore, they are incorrectly labeled as positive. This is actually related to Flipping Output [4], an approach for ensemble diversity enhancement.

Note that when transformed feature vectors are too long to be accommodated, feature sampling can be performed, e.g., by subsampling the instances generated by sliding window scanning, as completely-random trees do not rely on feature split selection whereas random forests are quite insensitive to feature split selection. The feature sampling process is also related to Random Subspace [17], an approach for ensemble diversity enhancement.

Figure 4 shows only one size of sliding window. By using multiple sizes of sliding windows, multi-grained feature vectors will be generated, as shown in Figure 5.

4.3 Overall Procedure and Hyper-Parameters

Figure 5 summarizes the overall procedure of gcForest. Suppose the original input is of 400 raw features, and three window sizes are used for multi-grained scanning. For m training examples, a window with size of 100 features will generate a data set of 301 × m 100-dimensional training examples. These data will be used to train a completely-random forest and a random forest, each containing 500 trees. If there are three classes to be predicted, an 1,806-dimensional feature vector will be obtained as described in Section 4.1. The transformed training set will then be used to train the 1st-grade of cascade forest.

Similarly, sliding windows with sizes of 200 and 300 features will generate 1,206-dimensional and 606-dimensional feature vector, respectively, for each original training example. The transformed feature vectors, augmented with the class vector generated by the previous grade, will then be used to train the 2nd-grade and 3rd-grade of cascade forests, respectively. This procedure will be repeated till convergence of validation performance. In other words, the final model is actually a *cascade of cascades*, where each cascade consists of multiple levels each corresponding to a grain of scanning, e.g., the 1st cascade consists of Level 1_A to Level 1_C in Figure 5. For difficult tasks, more grains can be used when computational resource allows.

Given a test instance, the multi-grained scanning procedure will be gone through to get the corresponding transformed feature representation, and then the cascade will be gone through till the last level. The final prediction will be obtained by aggregating the four 3-dimensional class vectors at the last level, and the class with the maximum aggre-

Table 1. Summary of hyper-parameters and default settings of gcForest. Boldfont highlights hyper-parameters with relatively larger influence; "?" indicates default value unknown, or generally requiring different settings for different tasks.

Deep neural networks (e.g., convolutional neural networks)	gcForest
Type of activation functions:	Type of forests:
Sigmoid, ReLU, tanh, linear, etc.	Completely-random forest, random forest, etc.
Architecture configurations:	Forest in multi-grained scanning:
No. Hidden layers: ?	**No. Forests**: $\{2\}$
No. Nodes in hidden layer: ?	**No. Trees in each forest**: $\{500\}$
No. Feature maps: ?	Tree growth: till pure leaf, or reach depth 100
Kernel size: ?	**Sliding window size**: $\{\lfloor d/16 \rfloor, \lfloor d/8 \rfloor, \lfloor d/4 \rfloor\}$
Optimization configurations:	Forest in cascade:
Learning rate: ?	**No. Forests**: $\{8\}$
Dropout: {0.25/0.50}	**No. Trees in each forest**: $\{500\}$
Momentum: ?	Tree growth: till pure leaf
L1/L2 weight regularization penalty: ?	
Weight initialization: Uniform, glorot_normal, glorot_uni, etc.	
Batch size: {32/64/128}	

gated value will be output.

Table 1 summarizes the hyper-parameters of typical DNNs and gcForest, where the default values of gcForest used in our experiments are given.

5 Experiments

5.1 Configuration

We compare gcForest with deep neural networks and several other popular learning algorithms. The implementations are based on Python, with neural neworks from Keras and traditional learning algorithms from Sklearn.

In all experiments gcForest is using the *same* cascade structure: Each level consists of 4 completely-random forests and 4 random forests, each containing 500 trees, as described in Section 4.1. Three-fold cross validation is used for class vector generation. The number of cascade levels is automatically determined. In detail, we split the training set into two parts, i.e., growing set and estimating set[1]; then we use the growing set to grow the cascade, and the estimating set to estimate the performance. If growing a new level does not improve the performance, the growth of the cascade terminates and the estimated number of levels is obtained. Then, the cascade is retrained based on merging the growing and estimating sets. For all experiments we take 80% of the training data for growing set and 20% for estimating set. For multi-grained scanning, three window sizes are used. For d raw features, we use feature windows with sizes of $\lfloor d/16 \rfloor$, $\lfloor d/8 \rfloor$, $\lfloor d/4 \rfloor$; if the raw features are with panel structure (such as images), the feature windows are also with panel structure as shown in Figure 4. Note that a careful task-specific tuning will lead to better performance; here, to highlight that the hyper-parameter setting of gcForest is much easier than deep neural networks, we simply use the same setting for all tasks, whereas task-specific tunings are conducted for DNNs.

For deep neural network configurations, we use ReLU for activation function, cross-entropy for loss function, adadelta for optimization, dropout rate 0.25 or 0.5 for hidden layers according to the scale of training data. The network structure hyper-parameters, however, cannot be fixed across tasks, otherwise the performance will be embarrassingly unsatisfactory. For example, a network attained 80% accuracy on ADULT dataset achieved only 30% accuracy on YEAST with the same architecture (only the number of input/output nodes changed to suit the data). Therefore, for DNNs, we examine a variety of architectures on validation set and pick the one with the best performance, then re-train the network on training set and report the test accuracy.

5.2 Results

We run experiments on a broad range of tasks, with data types of image, audio, time-series, text, etc.

Image Categorization

The MNIST dataset [25] contains 60,000 images of size 28×28 for training (and validating), and 10,000 images

[1]Some experimental datasets are given with training/validation sets. To avoid confusion, here we call the subsets generated from training set as growing/estimating sets.

for testing. Deep Belief Nets in [16] attained accuracy of 98.75%, a re-implementation of LeNet-5 (a modern version of LeNet with dropout and ReLUs) attains accuracy of 99.05%, SVM (rbf kernel) 98.60%, Random Forest 96.80%, whereas gcForest attains 99.26% by simply using default settings in Table 1.

Face Recognition

The ORL dataset [37] contains 400 gray-scale facial images taken from 40 persons. We randomly choose 5/7/9 images per person for training, and report the test performance on remaining images. Note that a random guess will achieve 2.5% accuracy, since there are 40 possible outcomes. We compare with a CNN consisting of 2 conv-layers with 32 feature maps of 3×3 kernel, and each conv-layer has a 2×2 max-pooling layer followed. A dense layer of 128 hidden units is fully connected with the convolutional layers and finally a fully connected soft-max layer with 40 hidden units is appended at the end. ReLU, cross-entropy loss, dropout rate of 0.25 and adadelta are used for training. The batch size is set to 10, and 50 epochs are used. We have also tried other configurations of CNN, but this one gives the best performance: test accuracy of 86.50%/91.67%/95.00% corresponding to 5/7/9 training images per person. The kNN ($k = 3$) test accuracies are 76.00%/83.33%/92.50%, SVM (rbf kernel) 80.50%/82.50%/85.00%, Random Forest 91.00%/93.33%/95.00%, whereas gcForest attains 91.00%/96.67%/97.50% by simply using default settings.

Music Classification

The GTZAN dataset [42] contains 10 genres of music clips, each represented by 100 tracks of 30 seconds long. We split the dataset into 700 clips for training and 300 clips for testing. In addition, we use MFCC feature to represent each 30 seconds music clip, which transforms the original sound wave into an $1,280 \times 13$ feature matrix. Each frame is atomic according to its own nature; thus, CNN uses a 13×8 kernel with 32 feature maps as the conv-layer, each followed by a pooling layer. Two fully connected layers with 1,024 and 512 units, respectively, are appended, and finally a soft-max layer is added in the last. We also compare it with an MLP having two hidden layers, with 1,024 and 512 units, respectively. Both networks use ReLU as activation function and categorical cross-entropy as the loss function. For Random Forest, Logistic Regression and SVM, each input is concatenated into an $1,280 \times 13$ feature vector. The test accuracies are: CNN 59.20%, MLP 58.00%, Random Forest 50.33%, Logistic Regression 50.00%, SVM (rbf kernel) 18.33%, whereas gcForest attains 65.67% by simply using default settings.

Hand Movement Recognition

The sEMG dataset [38] consists of 1,800 records each belonging to one of six hand movements, i.e., spherical, tip, palmar, lateral, cylindrical and hook. This is a time-series dataset, where EMG sensors capture 500 features per second and each record associated with 3,000 features. In addition to an MLP with *input*-1,024-512-*output* structure, we also evaluate a recurrent neural network, LSTM [11] with 128 hidden units and sequence length of 6 (500-dim input vector per second). The test accuracies are: LSTM 45.37%, MLP 38.52%, Random Forest 29.62%, SVM (rbf kernel) 29.62%, Logistic Regression 23.33%, whereas gcForest attains 71.30% by simply using default settings.

Sentiment Classification

The IMDB dataset [31] contains 25,000 movie reviews for training and 25,000 for testing. The reviews are text data represented by *tf-idf* features. This is not image data, and thus CNNs are not directly applicable; CNNs facilitated with word embedding achieved test accuracy 89.02% [19]. MLP with structure *input*-1,024-1,024-512-256-*output* attains 88.04%, SVM (rbf kernel) 87.56%, Random Forest 85.32%, Logistic Regression 88.62%. Considering that *tf-idf* features do not convey spacial or sequential relationships, we use default setting for gcForest but skip multi-grained scanning, and achieve the test accuracy 89.16%, even better than CNN facilitated with word embedding.

Low-Dimensional Data

We also evaluate gcForest on UCI-datasets [27] with relatively small number of features: LETTER with 16 features and 16,000/4,000 training/test examples, ADULT with 14 features and 32,561/16,281 training/test examples, and YEAST with only 8 features and 1,038/446 training/test examples. Fancy architectures like CNNs could not work on such data as there are too few features without spatial relationship. So, we compare with MLPs. Unfortunately, although MLPs have less configuration options than CNNs, they are still very tricky to set up. For example, MLP with *input*-16-8-8-*output* structure and ReLU activation achieves 76.37% accuracy on ADULT but just 33% on LETTER. We conclude that there is no way to have one MLP structure which gives decent performance across all datasets. Therefore, we report different MLP structures with the best performance: for LETTER the structure is *input*-70-50-*output* with test accuracy 95.70%, for ADULT is *input*-30-20-*output* with test accuracy 85.25%, and for YEAST is *input*-50-30-*output* with test accuracy 55.60%. These results are inferior to Random Forest: 96.50% on LETTER, 85.49% on ADULT, 61.66% on YEAST. In contrast, gcForest achieves 97.40% on LETTER, 86.40% on

ADULT, 63.45% on YEAST, by simply using default setting and abandoning multi-grained scanning by considering that the features of these small-scale data do not hold spacial/sequential relationships.

High-Dimensional Data

The CIFAR-10 dataset [23] contains 50,000 images of 10 classes for training and 10,000 images for testing. Here, each image is a 32×32 colored image with 8 gray-levels; thus, each instance is of 8,192-dim. ResNet achieved test accuracy 93.57% [14], AlexNet 83.00% [22], Deep Belief Net 62.20% [23], MLP 42.20% [1]. The test accuracies of non-DNN approaches are: Random Forest 50.17%, Logistic Regression 37.32%, SVM (linear kernel) 16.32%.

As we discussed in Section 4, currently we include only 10-dim augmented feature vector from each forest, and such a small number of augmented features will be easily drown out by the original long feature vector. Nevertheless, though the test accuracy of gcForest with default setting, 61.78%, is inferior to state-of-the-art DNNs, it is already the best among non-DNN approaches. Moreover, the performance of gcForest can be further improved via task-specific tuning, e.g., by including more grains (i.e., using more sliding window sizes in multi-grained scanning) like gcForest(5grains) which uses five grains and attains 63.37%. It is also interesting to see that the performance gets significant improvement to 69.00% with gcForest(gbdt), which simply replaces the final level with GBDT [6]. Section 5.8 will show that better performance can be obtained if larger models of gcForest can be trained.

5.3 Running time

Our experiments use a PC with 2 Intel E5 2695 v4 CPUs (18 cores), and the running efficiency of gcForest is good. For example, for IMDB dataset (25,000 examples with 5,000 features), it takes 267.1 seconds per cascade level, and automatically terminates with 9 cascade levels, amounting to 2,404 seconds or 40 minutes. In contrast, MLP compared on the same dataset requires 50 epochs for convergence and 93 seconds per epoch, amounting to 4,650 seconds or 77.5 minutes for training; 14 seconds per epoch (with batch size of 32) if using GPU (Nvidia Titan X pascal), amounting to 700 seconds or 11.6 minutes. Multi-grained scanning will increase the cost of gcForest; however, the different grains of scanning are inherently parallel. Also, both completely-random forest and random forest are parallel ensemble methods [50]. Thus, the efficiency of gcForest can be improved further with optimized parallel implementation. Note that the training cost is controllable because users can set the number of grains, forests, trees by considering computational cost available. It is also noteworthy that the above comparison is somewhat unfair to gcForest, because many different architectures have been tried for neural networks to achieve the reported performance but these time costs are not included.

5.4 Performance Tendency

Figure 6 shows the performance tendency of gcForest when the number of cascade level increases. It can be seen that gcForest starts from performance inferior to SVM and MLP, and gradually improves. In our experiments the validation process terminates the growth at the 9th level; the figure shows more levels for observing the tendency.

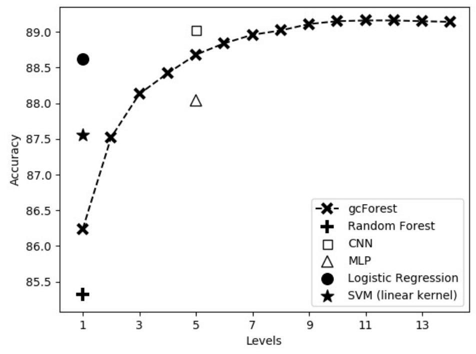


Figure 6. Performance tendency on IMDB.

5.5 Influence of Multi-Grained Scanning

To study the separate contribution of the cascade forest structure and multi-grained scanning, we compare gcForest with cascade forest on MNIST, GTZAN and sEMG datasets. The test accuracies with/without multi-grained scanning are 99.26%/65.67%/71.30% and 98.02%/52.33%/48.15% on MNIST/GTZAN/sEMG, respectively. It is evident that when there are spacial or sequential feature relationships, multi-grained scanning helps improve performance.

5.6 Influence of Completely-Random Forest

To study the contribution of completely-random forest, we compare gcForest with its variant which replaces completely-random forests by random forests. The test accuracies with/without completely-random forests are 99.26%/89.16%/97.40% and 99.11%/87.62%/96.65% on MNIST/IMDB/LETTER, respectively. It shows that completely-random forest helps no matter whether multi-grained scanning applied (MNIST) or not (IMDB), or whether data are low-dimensional (LETTER).

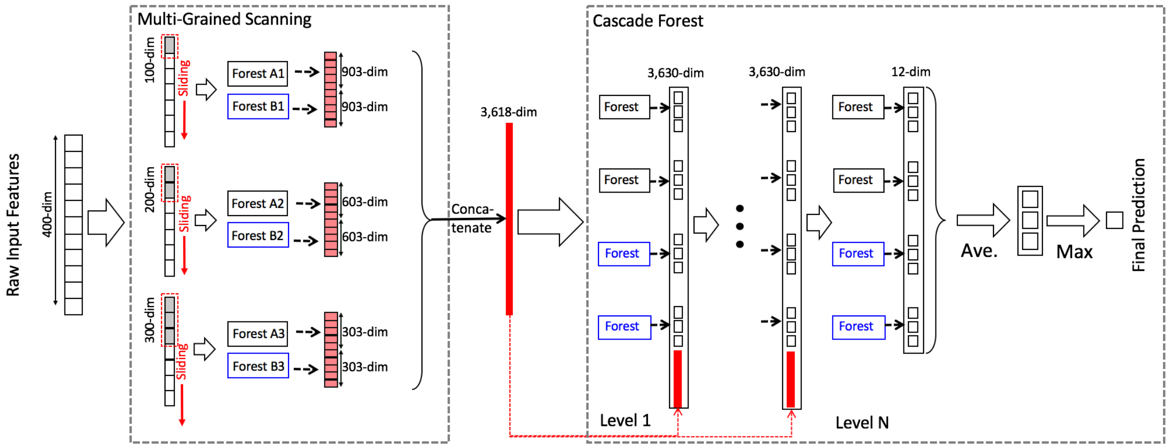


Figure 7. A variant which concatenates all features from multiple grains. Suppose there are three classes, raw features are 400-dim, and three sizes of sliding windows are used.

5.7 Influence of Cascade Structure

The final model structure of gcForest is a *cascade of cascades*, where each cascade consists of multiple levels each corresponding to a grain of scanning, as shown in Figure 5. There are other possible ways to exploit the features from multiple grains, e.g., the variant which concatenates all features together as shown in Figure 7.

Table 2 shows that concatenating the features from multiple grains is not as good as the current design in gcForest (here, ORL is with 9 training images per person). Nevertheless, there might be other ways leading to better results; we leave it for future exploration.

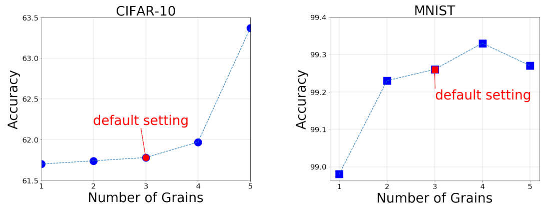
Table 2. Results with the variant of concatenating features from multiple grains.

	MNIST	ORL	GTZAN	sEMG
gcForest	**99.26%**	97.50%	65.67%	**71.30%**
variant	98.96%	**98.30%**	65.67%	55.93%
	IMDB	LETTER	ADULT	YEAST
gcForest	89.16%	**97.40%**	**86.40%**	**63.45%**
variant	**89.32%**	97.25%	86.17%	63.23%

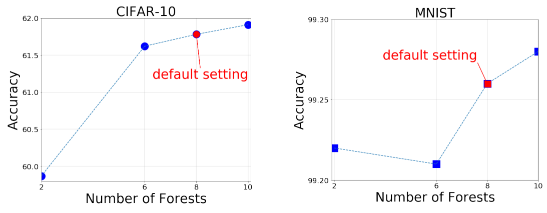
5.8 Influence of Larger Models

Figure 8 suggests that larger models tend to offer better performance, though we have not tried even more grains, forests and trees due to limit of computational resource.

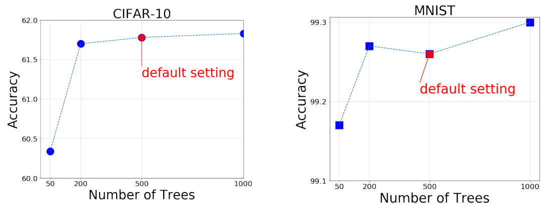
Note that computational facilities are crucial for enabling the training of larger models; e.g., GPUs for DNNs. On one hand, some new computational devices, such as Intel KNL of the MIC (Many Integrated Core) architecture, may offer



(a) With increasing number of grains.



(b) With increasing number of forests per grade.



(c) With increasing number of trees per forest.

Figure 8. Performance with increasing number of grains/forests/trees. Red color highlights the performance with default setting.

acceleration for gcForest. On the other hand, some components of gcForest, e.g., multi-grained scanning, may be accelerated by GPUs. Moreover, there is plenty of room for improvement with distributed computing implementations.

6 Real Application

The gcForest has been implemented in an industrial distributed machine learning platform and applied to real-world illegal cash-out fraud detection by a big unicorn enterprise [49]. On dataset with 131,407,704 training examples and 52,489,529 testing examples each corresponding to a transaction described by 5,000 features, gcForest achieved the best performance 0.9970/0.5440/0.9480 on AUC/F1/KS, whereas DNNs achieved 0.9722/0.3861/0.8551. Details see [49].

7 Related Work

The gcForest is a decision tree ensemble approach. There are studies showing that by using ensembles such as random forest facilitated with DNN features, the performance can be even better than simply using DNNs [21]. Our purpose is quite different. We are aiming at a non-NN style deep model rather than a combination with DNNs. By using cascade forest structure, we hope to endow the model with characteristics of layer-by-layer processing, in-model feature transformation and sufficient model complexity.

Random forest [5], which has been widely applied to various tasks, is one of the most successful ensemble methods. Completely-random forest has been found useful during recent years, such as iForest [29] for anomaly detection, sencForest [34] for handling emerging new classes in streaming data, etc. The gcForest offers another example exhibiting the usefulness of completely-random forest.

Many works tried to connect random forest with neural networks, such as converting cascaded random forest to CNNs [36] and exploiting random forest to help initialize neural networks [46]. These works are typically based on early studies, e.g., mapping of trees to networks,tree-structured neural networks,as reviewed in [51]. Their goals are totally different from ours. In particular, their final models are based on differentiable modules (even for studies involving non-differentiable activation functions, differentiable relaxation functions are actually used in optimization/learning process), whereas we are trying to develop deep models based on non-differentiable modules without relying on gradient-based adjustment.

The multi-grained scanning procedure of gcForest uses different sizes of sliding windows to examine the data; this is somewhat related to wavelet and other multi-resolution examination procedures [32]. For each window size, a set of instances are generated from one training example; this is related to bag generators of multi-instance learning [44]. In particular, the bottom part of Figure 4, if applied to images, can be regarded as the *SB* image bag generator [44].

The cascade procedure of gcForest is related to Boosting [10], which is able to automatically decide the number of learners in ensemble, and particularly, a cascade boosting procedure [43] has achieved great success in object detection tasks. Note that when multiple grains are used, each level in the cascade of gcForest consists of multiple grades; this is actually a cascade of cascades. Each grade can be regarded as an ensemble of ensembles.

Passing outputs of one level of learners as inputs to another level of learners is related to stacking [3, 47]. However, stacking is easy to overfit with more than two levels, and can hardly enable a deep model by itself. Our trick lies in the enhancement of diversity during model growth. Actually, gcForest exploits all major categories of diversity enhancement strategies [50].

As a tree-based approach, gcForest might be potentially amicable for theoretical analysis than deep neural networks, although this is beyond the scope of this paper. Indeed, some recent theoretical studies about deep learning, e.g., [33], seem more intimate with tree-based models.

8 Future Issues

One important future issue is to enhance the feature re-representation process. The current implementation of gcForest takes the simplest form of class vectors, i.e., the class distribution at the leaf nodes into which the concerned instance falls. Such a small number of augmented features will be easily drown out when the original feature vectors are high-dimensional. It is apparent that more features may be involved. Intuitively, more features may enable the incorporation of more information, although not always necessarily helpful for generalization. Moreover, a longer class vector may enable a joint multi-grained scanning process, leading to more flexibility of re-representation. Recently we show that decision forest can serve as AutoEncoder [9]. On one hand, this shows that the ability of AutoEncoder is not a special property of neural networks; on the other hand, this shows that a forest can encode abundant information, offering great potential to facilitate feature re-representation.

Another important future issue is to accelerate and reduce the memory consumption. Building larger models may lead to better performance, where computational facilities are crucial for enabling the training of larger models. The success of DNNs owes much to the acceleration offered by GPUs, but forest structure is unfortunately not suitable to GPUs. One possibility is to consider some new computational devices such as KNL; another is distributed computing implementation. Feature sampling can be executed when transformed feature vectors produced by multi-grained scanning are too long to be accommodated; this not only helps storage reduction, but also offers another channel to enhance the diversity. It is also possible to explore smarter sampling strategies such as BLB [20] or feature hashing [45] when adequate. The *hard negative mining* strategy may help improve generalization, and effort improving the efficiency of hard negative mining may also be helpful for multi-grained scanning [15]. The efficiency of gcForest may be further improved by reusing some components during the process of different grained scanning, class vectors generation, forest training, completely-random trees generation, etc. In case the learned model is big, it is possible to reduce to a smaller one by using the strategy presented in [53], later called knowledge distillation.

The adoption of completely-random forest not only helps diversity enhancement, but also provides an opportunity to exploit unlabeled data. Note that the growth of completely-random trees does not require labels, whereas label information is only needed for annotating leaf nodes. Intuitively, for each leaf node it might be able to require only one labeled example if the node is to be annotated according to the majority cluster on the node, or one labeled example per cluster if all clusters in the node are innegligible. This offers gcForest with the opportunity of incorporating active learning [18] and/or semi-supervised learning strategies [54].

The gcForest is able to achieve performance highly competitive to DNNs on a broad range of tasks except some large-scale image task. Indeed DNNs are very successful on image tasks, e.g., [26, 48]. On one hand, we believe that the performance of gcForest can be significantly improved, e.g., by designing better feature re-representation scheme rather than using the current simple classification vectors. On the other hand, it should not be ignored that DNN models have been investigated for more than twenty years by huge crowd of researchers/engineers whereas deep forest is just born. Furthermore, we conjecture that numeric modeling tasks such as image/audio data are very suitable to DNNs because their operations, such as convolution, fit well with numeric signal modeling. Deep forest is not developed to replace DNNs for such tasks; instead, it offers an alternative when DNNs are not superior. There are plenty of tasks, especially categorical/symbolic or mixed modeling tasks, where deep forest may be found useful. For example, the application described in Section 6 is a mixed modeling task involving both categorical and numeric features.

9 Conclusion

This paper attempts to address the question that *Can deep learning be realized with non-differentiable modules?* We conjecture that behind the mystery of deep neural networks there are three crucial characteristics, i.e., layer-by-layer processing, in-model feature transformation, and sufficient model complexity. To verify the conjecture, we try to endow these characteristics to a non-NN style deep model, and our results show that it really works.

The proposed gcForest approach[2] is able to construct *deep forest*, a deep model based on decision trees, and the training process does not rely on backpropagation and gradient adjustment. Comparing with deep neural networks, gcForest has less hyper-parameters and achieved excellent performance across various domains by using even the same parameter setting.

There are other possibilities to construct deep forest. As a seminal study, we have only explored a little in this direction. Indeed, the most important value of this paper lies in the fact that it may open a door for non-NN style deep learning, or deep models based on non-differentiable modules that does not rely on gradients.

Funding

This research has been supported by the National Science Foundation of China (61751306) and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] J. Ba and R. Caruana. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2654–2662. 2014.

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

[3] L. Breiman. Stacked regressions. *Machine Learning*, 24(1):49–64, 1996.

[4] L. Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):113–120, 2000.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 785–794, San Francisco, CA, 2016.

[7] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3123–3131. 2015.

[8] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[2]A shared code of gcForest for small or middle scale data is available at http://lamda.nju.edu.cn/code_gcForest.ashx.

[9] J. Feng and Z.-H. Zhou. AutoEncoder by forest. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, New Orleans, LA, 2018.

[10] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[11] F. A. Gers, D. Eck, and J. Schmidhuber. Applying LSTM to time series predictable through time-window approaches. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 669–676, Vienna, Austria, 2001.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[13] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 770–778, Las Vegas, NV, 2016.

[15] J. F. Henriques, J. Carreira, R. Caseiro, and J. Batista. Beyond hard negative mining: Efficient detector learning via block-circulant decomposition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2760–2767, Sydney, Australia, 2013.

[16] G. E. Hinton, S. Osindero, and Y.-W. Simon. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[17] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[18] S.-J. Huang, R. Jin, and Z.-H. Zhou. Active learning by querying informative and representative examples. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 36(10):1936–1949, 2014.

[19] Y. Kim. Convolutional neural networks for sentence classification. *arXiv:1408.5882*, 2014.

[20] A. Kleiner, A. Talwalkar, F. Sarkar, and M. I. Jordan. The big data bootstrap. In *Proceedings of the 29th International Conference on Machine Learning*, Edinburgh, Scotland, 2012.

[21] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1467–1475, Santiago, Chile, 2015.

[22] A. Krizhenvsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[23] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[24] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 231–238. 1995.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] Y. Li, Y. Zhang, X. Huang, H. Zhu, and J. Ma. Large-scale remote sensing image retrieval by deep hashing neural networks. *IEEE Trans. Geoscience and Remote Sensing*, 56(2):950–965, 2018.

[27] M. Lichman. UCI machine learning repository, 2013.

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

[29] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining*, pages 413–422, Pisa, Italy, 2008.

[30] J.-H. Luo, J. Wu, and W. Lin. ThiNet: A filter level pruning method for deep neural network compression. *arXiv:1707.06342*, 2017.

[31] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 142–150, Portland, OR, 2011.

[32] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, London, UK, 2nd edition, 1999.

[33] H. Mhaskar, Q. Liao, and T. A. Poggio. When and why are deep networks better than shallow ones? In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 2343–2349, San Francisco, CA, 2017.

[34] X. Mu, K. M. Ting, and Z.-H. Zhou. Classification under streaming emerging new classes: A solution using completely-random trees. *IEEE Trans. Knowledge and Data Engineering*, 29(8):1605–1618, 2017.

[35] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proceedings of the 14th European Conference on Computer Vision*, pages 525–542, Amsterdam, The Netherlands, 2016.

[36] D. L. Richmond, D. Kainmueller, M. Y. Yang, E. W. Myers, and C. Rother. Relating cascaded random forests to deep convolutional neural networks for semantic segmentation. *arXiv:1507.07583*, 2015.

[37] F. Samaria and A. C. Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of the 2nd IEEE Workshop on Applications of Computer Vision*, pages 138–142, Sarasota, FL, 1994.

[38] C. Sapsanis, G. Georgoulas, A. Tzes, and D. Lymberopoulos. Improving EMG based classification of basic hand movements using EMD. In *Proceedings of the 35th Annual International Conference on the IEEE Engineering in Medicine and Biology Society*, pages 5754–5757, Osaka, Japan, 2013.

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

[40] J. Sirignano. Deep learning models in finance. *SIAM News*, 50(5):1, 2017.

[41] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2377–2385. 2015.

[42] G. Tzanetakis and P. R. Cook. Musical genre classification of audio signals. *IEEE Trans. Speech and Audio Processing*, 10(5):293–302, 2002.

[43] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 511–518, Kauai, HI, 2001.

[44] X.-S. Wei and Z.-H. Zhou. An empirical study on image bag generators for multi-instance learning. *Machine Learning*, 105(2):155–198, 2016.

[45] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1113–1120, Montreal, Canada, 2009.

[46] J. Welbl. Casting random forests as artificial neural networks (and profiting from it). In *Proceedings of the 36th German Conference on Pattern Recognition*, pages 765–771, Münster, Germany, 2014.

[47] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–260, 1992.

[48] T. Xu, X. Zhang, F. Liu, and L. Jiao. Unsupervised deep feature learning for remote sensing image retrieval. *Remote Sensing*, 10(8):1243, 2018.

[49] 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. *arXiv:1805.04234*, 2018.

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

[51] Z.-H. Zhou and Z.-Q. Chen. Hybrid decision trees. *Knowledge-Based Systems*, 15(8):515–528, 2002.

[52] Z.-H. Zhou and J. Feng. Deep forest: Towards an alternative to deep neural networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3553–3559, Melbourne, Australia, 2017.

[53] Z.-H. Zhou and Y. Jiang. NeC4.5: Neural ensemble based C4.5. *IEEE Trans. Knowledge and Data Engineering*, 16(6):770–773, 2004.

[54] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knowledge and Information Systems*, 24(3):415–439, 2010.