Multi-layer heterogeneous ensemble with classifier and feature selection.

NGUYEN, T.T., VAN PHAM, N., DANG, M.T., LUONG, A.V., MCCALL, J. and LIEW, A.W.C.

2020

Multi-layer Heterogeneous Ensemble with Classifier and Feature Selection

Tien Thanh Nguyen
School of Computing Science and Digital Media, Robert Gordon University
Aberdeen, UK

Nang Van Pham
School of Electrical Engineering, Hanoi University of Science and Technology
Vietnam

Manh Truong Dang
School of Computing Science and Digital Media, Robert Gordon University
Aberdeen, UK

Anh Vu Luong
School of Information and Communication Technology, Griffith University
Australia

John McCall
School of Computing Science and Digital Media, Robert Gordon University
Aberdeen, UK

Alan Wee Chung Liew
School of Information and Communication Technology, Griffith University
Australia

ABSTRACT

Deep Neural Networks have achieved many successes when applying to visual, text, and speech information in various domains. The crucial reasons behind these successes are the multi-layer architecture and the in-model feature transformation of deep learning models. These design principles have inspired other sub-fields of machine learning including ensemble learning. In recent years, there are some deep homogenous ensemble models introduced with a large number of classifiers in each layer. These models, thus, require a costly computational classification. Moreover, the existing deep ensemble models use all classifiers including unnecessary ones which can reduce the predictive accuracy of the ensemble. In this study, we propose a multi-layer ensemble learning framework called MUlti-Layer heterogeneous Ensemble System (MULES) to solve the classification problem. The proposed system works with a small number of heterogeneous classifiers to obtain ensemble diversity, therefore being efficiency in resource usage. We also propose an Evolutionary Algorithm-based selection method to select the subset of suitable classifiers and features at each layer to enhance the predictive performance of MULES. The selection method uses NSGA-II algorithm to optimize two objectives concerning classification accuracy and ensemble diversity. Experiments on 33 datasets confirm that MULES is better than a number of well-known benchmark algorithms.

CCS CONCEPTS

• **Computing methodologies** → **Ensemble methods**; • **Mathematics of computing** → Evolutionary Algorithms;

KEYWORDS

Ensemble method, deep learning, multiple classifiers, ensemble of classifiers, feature selection, classifier selection

GECCO '20, July 8–12, 2020, Cancún, Mexico

1 INTRODUCTION

In recent years "deep learning" has become one of the most popular terms in the research community. Deep Neural Networks (DNNs) have shown outstanding achievements in many supervised learning tasks for visual, text, and speech information [13]. In computer vision, for instance, Convolutional Neural Network (CNN), a DNN, significantly outperforms traditional machine learning algorithms on the large-scale ImageNet classification task [11]. Zhou and Feng summarized three crucial reasons for the success of DNNs, including layer-by-layer processing, in-model feature transformation, and sufficient model complexity [30]. From the training data, DNNs generates the new training input at each layer in which the new training features reflect the different high-level abstract representations of the original data. This multi-layer design brings out different aspects of the original data in comparison to the flat networks i.e. single-hidden layer networks or traditional machine learning models which only work on the original data. Moreover, DNNs are designed with many hyper-parameters in many layers, resulting in high model complexity. Such high complexity is needed to explore the large training data.

DNNs are built based on neural networks by using backpropagation technique to train multi-layer with differentiable nonlinear modules. Recently, some efforts are being done to extend deep learning models with many layers of ensemble of classifiers [3, 21, 24, 30]. An ensemble is a collection of classifiers whose prediction are combined so as to achieve better performance than its constituent members. The key success of ensemble learning results from the diverse exploration of the original data using different hypothesis i.e. classifiers. By constructing deep ensemble models, we expect to

benefit from both the deep learning and ensemble learning mechanisms. Although these deep ensemble models achieve superior classification performance on many datasets, the existing ensemble requires a large number of classifiers, resulting in huge memory usage and expensive computation. Moreover, these deep ensemble models do not perform classifier selection for each layer. In fact, the presence of some poor classifiers in each layer could degrade the system performance. We addressed these limitations by designing a multi-layer ensemble model with ensemble selection.

This paper aims to design an effective ensemble-based supervised learning system for classification inspired by the representation learning using layer-by-layer processing of DNNs. To achieve this goal, the objectives below have been specified:

- Design a novel ensemble model involving a multi-layer architecture with diverse classifiers in each layer (called MULES). By combining the advantages of ensemble learning and deep learning, the proposed system is expected to perform well in many classification tasks.
- Investigate an approach to simultaneously select classifiers and the features in each layer. The proposed approach thus can overcome the limitation of existing deep ensemble models by selecting only the subset of suitable classifiers and features for each layer.
- Propose fitness measures for MULES by considering not only classification accuracy but also the ensemble diversity generated by classifiers in each layer.
- Compare the performance of MULES to several existing ensemble methods and deep ensemble models.

In Section 2, we briefly introduce ensemble learning for classification and some recent developments in Evolutionary Algorithms for ensemble systems. In Section 3, we give a detail description of the general architecture for the MULES. Experimental studies on a number of datasets are provided in Section 4, followed by conclusions in Section 5.

2 BACKGROUND AND RELATED WORK

2.1 Ensemble Learning

Ensemble learning explores the original data by using different hypothesis, i.e. classifiers. There are two strategies to generate the set of classifiers: heterogeneity or homogeneity [2, 4, 19]. In homogeneous ensemble, many classifiers are obtained by training one learning algorithm on many different training sets obtained from the original one. Homogeneous ensemble uses many classifiers to ensure that the classification error converges to its asymptotic value. This thus requires huge memory storage for classifiers and high computational budgets for classification. Among many homogeneous ensemble frameworks, Random Forest and XgBoost are the top-performance methods [4, 8]. The heterogeneous ensemble, in contrast, uses a small number of different learning algorithms on the training data to generate diverse classifiers. The research on heterogeneous ensemble focuses on designing combining algorithms that effectively combine the predictions of the classifiers. Some examples of combining algorithms are Bayesian-based method with Mixture of Gaussians [16] and Information Granularity-based method [19].

Ensemble methods based on a multi-layer architecture is nowadays becoming a popular trend in ensemble system design. Such systems involve more than one layer of ensembles. Some examples of two layers of ensembles are two-layer heterogeneous ensemble [17], and a two-layer ensemble of Random Subspace and Bagging [29]. Viola and John [25] proposed a cascade model as a sequence of binary classifiers. If one classifier outputs positive value, the data is transmitted to the next classifier. A deep ensemble learning model with more than two layers called gcForest was introduced by Zhou and Feng [30] with four random forests in each layer. Utkin et al. [24] optimized gcForest by considering the weights of the trees in the same forest when averaging their predictions. These weights are found by minimizing a loss function on the training data based on the Euclidean distance between the weighted average vector and the crisp distribution vector based on the class labels of the training instances. Qi et al. [21] introduced a deep model with ensemble of SVM classifiers in each layer. The parameters of the models including the kernel functions of SVM classifiers, the number of classifiers, and the weights of features are found by using AdaBoost. In deep ensemble model in [3], the training input for i^{th} layer is formed by concatenating the original training data and all predictions of classifiers from the 1^{th} layer to the $(i-1)^{th}$ layer. The optimal hyper-parameters of the proposed model, including the number of classifiers in each layer, the number of layers, and the parameters for classifiers in each layer, are found by an Evolutionary Algorithm.

2.2 Selection Problem with Evolutionary Algorithms

In the past years, many applications of Evolutionary Algorithm (EA) have been proposed to select the optimal subset of classifiers or features to improve the ensemble performance. Nguyen et al. [15] proposed the encoding of both the classifiers and six combining methods in a chromosome. The final optimal set of combining methods is combined once again using the OWA operator. Chen et al. [5] used Ant Colony Optimization (ACO) to find the optimal set of classifiers in the ensemble system. A combining algorithm is chosen from a given set of algorithms by using the uniform distribution. Wang et al. [26] used NSGA-II [6] to search for the optimal set of classifiers generated by training a regression tree on 100 new training sets. The new training data was obtained by applying the random subspace and bootstrap resampling techniques on the original training set.

EA is also widely applied to feature selection which aims to remove redundant and irrelevant attributes to improve predictive performance and efficiency [28]. In ensemble learning, some EA-based feature selection methods have been proposed to select features for classifiers or to select predictions of classifiers for the combining algorithms. Kim and Cho [9] encoded the feature selection methods that will be used for each learning algorithm to learn the classifiers. The optimal solution is then obtained by using GA. In their extended version, the chromosomes in GA encodes the weight for each feature-classifier pair for the weighted sum combining rule. Bacauskiene et al. [1] built the homogeneous ensemble system of SVM classifiers in which the parameters of each classifier i.e. the regularization constant and the kernel width and the features used by each classifier are encoded via the binary encoding scheme. Nguyen et al. [18] used ACO to simultaneously select the optimum from the

predictions of classifiers and the set of combining algorithms to improve ensemble performance.

Recently, some Evolutionary Algorithms-based approaches have been proposed for DNNs optimization in terms of their topology and hyper-parameters [14]. In [27], Genetic Algorithm was used to optimize the network structures of CNNs. The CNN network was organized in several states including ordered nodes. A binary encoding scheme was used in each stage to encode the connections between the nodes insides. In [23], the three different building blocks: the convolutional layer, the pooling layer, and the fully connected layers in a CNN were encoded in one chromosome for evolution. The individuals are organized in variable-length encoding so as to reach the optimal depth of architecture. In [10], the hyper-parameters of DenseNet, a CNN, including the number of blocks, the number of layers in each block, and the growth rate was optimized by considering two objectives, namely computation accuracy and computational cost. The taxonomy of Evolutionary Algorithms to optimize DNNs can be found in [14].

3 PROPOSED SYSTEM

3.1 Deep Heterogeneous Ensemble System

Let $\mathcal{D} = \{(\mathbf{x_n}, \hat{y}_n)\}$, $|\mathcal{D}| = N$ be the training data, where $\mathbf{x_n} \in \mathcal{R}^D$ is the D-feature vector of the training instance, $\hat{y}_n \in \mathcal{Y} = \{y_m\}$, $|\mathcal{Y}| = M$ be its corresponding label that belongs to the label set with M labels, $\mathcal{K} = \{\mathcal{K}_k\}$ be the set of learning algorithms $|\mathcal{K}| = K$. In supervised learning (i.e. classification or regression), we aim to learn a hypothesis $\mathbf{h}$ (i.e., classifier) for the unknown relationship $g : \mathbf{x_n} \rightarrow \hat{y}_n$ and use this hypothesis to assign a label for each unlabeled instance. By using an ensemble system, we train an ensemble of classifiers (EoC) on $\mathcal{D}$ to obtain several different hypotheses $\{\widetilde{\mathbf{h_i}}\}$ for g. These hypotheses are then combined by a combining algorithm $C : \widetilde{\mathbf{h}} = C\{\widetilde{\mathbf{h_i}}\}$ for final decision making.

In this study, we propose MULES to solve the classification problems. The proposed system has multiple layers including different classifiers in each layer. The classifiers in the first layer train on the original training data and generate the new input training data for the second layer. The classifiers in the next layer train on the new training data generated by the preceding layer and generate the new training data for the subsequent layer. A layer thus can be viewed as the feature generator which generates features for the next layer. The combining algorithm then trains on the predictions of the classifiers in the last layer for collaborated prediction. In fact, this model is based on the idea of DNNs in which the data is passed through several layers in the learning system. However, it is different from DNNs since the information in the MULES is processed under the feed-forward mechanism in which the information is only passed from one layer to the next layer and does not involve back-propagation like in neural network methods. Moreover, several different learning algorithms will be used in each layer, whereas DNNs only work with layers of neurons. An illustration of MULES with a 3 layer-architecture and several different classifiers in each layer is shown in Figure 1. The outputs of the Decision Tree, Naïve Bayes, and Random Forest classifier in the first layer are combined with the original training data to generate the input for the second layer. A similar scheme is used at the second layer with two classifiers (SVM and Decision Tree) and in the third layer with two classifiers (LDA and SVM), before the output of the third layer is combined for the final prediction.

Two following questions arise from the proposed MULES:

- How to grow the deep model?
- How to combine the predictions of the last layer to obtain the final prediction?

To grow the deep model of MULES, it is needed to generate the input data for one layer. In this study, we concatenate the original training data with the predictions of classifiers of the i^{th} layer to form the new input data for the $(i+1)^{th}$ layer. We use T_i-fold cross-validation on the input data for the i^{th} layer to generate the predictions for the training data. In cross-validation, the input for i^{th} layer is divided into T_i disjoint parts in which the cardinality of each part is nearly similar. The predictions for observation in each part will obtain by using classifiers trained on the other parts. An observation thus will be predicted one time. We obtain the prediction of the k^{th} classifier at the i^{th} layer that $\mathbf{x_n}$ belongs to the class label y_m denoted by $p^{(i)}_{(k,m)}(\mathbf{x_n})$. We assume that the classifiers output the predictions in the form of probability [19]:

$$\sum_{m=1}^{M} p^{(i)}_{k,m}(\mathbf{x_n}) = 1; k = 1, ..., K; n = 1, ..., N \tag{1}$$

Let $\mathcal{L}_i$ denotes the new data generated by the i^{th} layer as the input for the $(i+1)^{th}$ layer ($i = 1, 2, \dots$) and $\mathcal{L}^{(0)} = \mathcal{D}$. In this study, we propose to generate $\mathcal{L}_i$ by concatenating the original training data and the prediction of each classifier for each observation. The augmented features, i.e. predictions and original features is expected to improve the discriminative characteristic of the training data [30]. In detail, the predictions for the training set at the i^{th} layer is given as a $N \times (MK)$ matrix $P^{(i)}(\mathbf{x_n}) = \left[p^{(i)}_{1,1}(\mathbf{x_n}), p^{(i)}_{1,2}(\mathbf{x_n}), ..., p^{(i)}_{K,M}(\mathbf{x_n})\right]$ while the original training data is given as a $N \times D$ matrix. The concatenation operator will generate a new input for the $(i+1)^{th}$ layer in the form of a $N \times (D + MK)$ matrix including the original training features and predictions of all instances arranged in the order as in Eqn. (3).

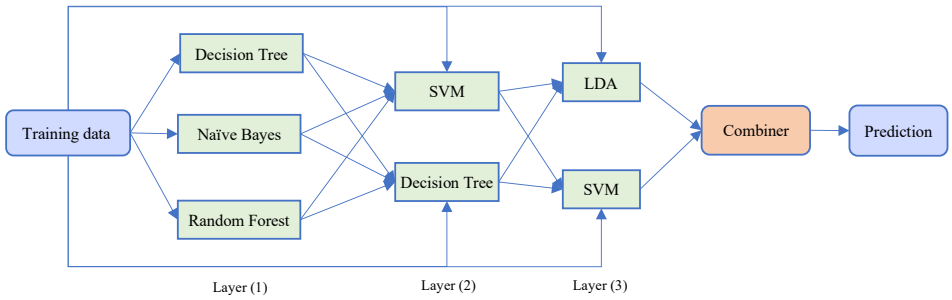
$$\mathcal{L}^{(i)} = \left[\mathcal{L}^{(i)}(\mathbf{x_1})...\mathcal{L}^{(i)}(\mathbf{x_N})\right]^T \tag{2}$$

$$\mathcal{L}^{(i)}(\mathbf{x_n}) = \left[x_{n1}, ..., x_{nD}, p^{(i)}_{1,1}(\mathbf{x_n}), p^{(i)}_{1,2}(\mathbf{x_n}), ..., p^{(i)}_{K,M}(\mathbf{x_n})\right] \tag{3}$$

In ensemble systems, an algorithm is used to combine the predictions of classifiers for the collaborated prediction. In this study, we used the Sum Rule method for combining [10]. Sum Rule summarizes the predictions of each instance with regard to each class label and assigns the instance to the class label associated with the maximum value. The combined prediction on an instance $\mathbf{x}$ at the i^{th} layer is given by:

$$\mathbf{x} \in y_t \ \text{ if } \ t = \text{argmax}_{m=1,...,M}\left\{\sum_{k=1}^{K} p^{(i)}_{k,m}(\mathbf{x})\right\} \tag{4}$$

The classification process works in a straightforward manner where a test instance $\mathbf{x}$ is fed forward through the layers to finally obtain



To keep the figure simple, we do not show the concatenation between training data and the prediction of each classifier to create the input for the next layer.

Figure 1: An illustration of the proposed deep heterogeneous ensemble system.

the predictions when reaching the output layer. Once again, the predictions of K classifiers at i^{th} layer for $\mathbf{x}$, i.e. $\mathcal{L}_i(\mathbf{x})$ are concatenated with the original training data in a shape of a $(D + MK)$-vector (3). The prediction vector of the last layer is fed into the combiner to generate the predicted label.

3.2 Simultaneous Classifier and Feature Selection Approach

At each layer of MULES, the original training data is concatenated to the predictions of classifiers to generate the input for the next layer. The prediction of a classifier can be viewed as scaled data from feature domain to probability domain. The new data in the probability domain may have better discriminative characteristics than the original training data in case of correct predictions where observations that belong to the same class will have similar prediction results and stay close together in the probability domain. However, in cases of wrong predictions, the discriminative characteristic of data will be mitigated. Concatenating the original training data and the classifier's predictions can therefore result in better or worse discriminative characteristics than the original training data. In this work, we develop a simultaneous classifier and feature selection method for each layer of MULES in which the system performance can be improved by either removing some inconsistent classifiers in each layer or selecting the most suitable features for each selected classifier when training on the input data.

We design a two-part encoding representation for the simultaneous classifier and feature selection method. The first part $[h_k^{(i)}, k = 1, ..., K]$ of the proposed encoding $E^{(i)}$ in Eqn. (5) is the encoding of the K classifiers and the second part $\left[f_{kd}^{(i)}, k = 1, ..., K; d = 1, ..., d_i\right]$ is the encoding of the features used by each classifier. In both parts, each gene in the encoding get two binary values {0,1} showing which classifiers is absence or presence in each layer (Eqn. (6)) or which features will be used by a classifier (Eqn. (7)).

$$E^{(i)} = \begin{bmatrix} h_1^{(i)} \dots h_K^{(i)} \\ f_{11}^{(i)}, f_{12}^{(i)}, \dots, f_{1d_i}^{(i)} \\ f_{21}^{(i)}, f_{22}^{(i)}, \dots, f_{2d_i}^{(i)} \\ \dots \\ f_{K1}^{(i)}, f_{K2}^{(i)}, \dots, f_{Kd_i}^{(i)} \end{bmatrix} \tag{5}$$

where d_i is the number of training features for layer i

$$h_k^{(i)} = \begin{cases} 1, & k^{th} \text{classifier is selected at layer } \ i \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$f_{kd}^{(i)} = \begin{cases} 1, & d^{th} \text{ feature is used by } k^{th} \text{ classifier at layer } \ i \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

In this study, the model selection is conducted at each layer to obtain the optimal set of classifiers and their features. The purpose of classifiers in all layers except the last one is to generate the input data for the next layer. Thus, the attention on only prediction accuracy is not enough to ensure the generation of effective input data. Here we consider bi-objective optimization for the model selection problem. The first objective is the accuracy of the classification task on the validation set $\mathcal{V}$:

$$\max_{E^{(i)}} \left\{ \frac{1}{|\mathcal{V}|} \sum_{n=1}^{|\mathcal{V}|} [\![\widetilde{\mathbf{h}}^{(i)}(\mathbf{x_n}) = \hat{y}_n]\!] \right\} \tag{8}$$

where $\widetilde{\mathbf{h}}^{(i)}$ is the combining model using Sum Rule at the i^{th} layer, $|\cdot|$ denotes the cardinality of a set, and $[\![\cdot]\!]$ is equal 1 if the condition is true, otherwise equal 0.

The second objective concerns the ensemble diversity generated by classifiers in each layer. It is widely recognized that diversity is an important factor to be considered when designing an ensemble system [12, 20, 30]. Kuncheva et al. [12] studied ten pairwise and non-pairwise diversity measures and examined the relationship between accuracy and diversity. Diversity was considered when designing an evaluation measure for ensemble selection (aka ensemble pruning) [20]. In this study, we measure the diversity of

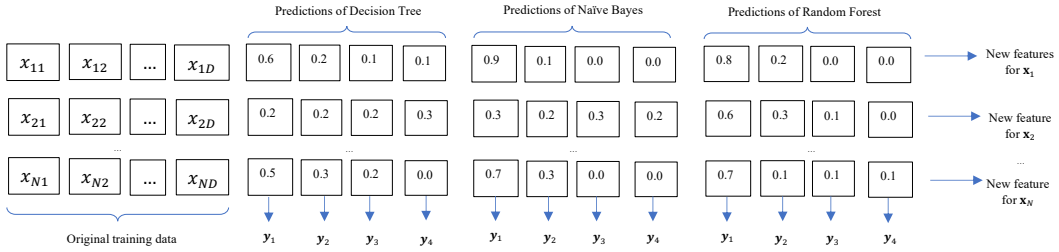


Figure 2 presents an example of the new training data generated by the first layer in Figure 1 for a four class-classification problem. Each of the three groups is the predictions of Decision Tree, Naïve Bayes, or Random Forest which show the probabilities that one instance belongs to the class labels. The new training features are concatenated with the original training data and predictions. For instances, the prediction vector (0.6, 0.2, 0.1, 0.1) in the first group shows the probabilities instance $\mathbf{x}_1$ *belongs to class label* y_1, y_2, y_3, *and* y_4 *respectively given by the Decision Tree classifier. The new training features of* $\mathbf{x}_1$ *is* (x_{11}, x_{12},..., x_{1D}, *0.6, 0.2, 0.1, 0.1, 0.9, 0.1, 0.0, 0.0, 0.8, 0.2, 0.0, 0.0) will be used as the input training data for the* 2^{nd} *layer.*

Figure 2: An example of the output data of the first layer in Figure 1.

classifiers in each layer by using Q-statistic computed on the predictive outcomes of two classifiers [12]:

$$Q_{ij} = \frac{N_{00}N_{11} - N_{10}N_{01}}{N_{00}N_{11} + N_{10}N_{01}} \tag{9}$$

in which N_{11} denotes the number of samples which are correctly classified by both classifiers, N_{10} denotes the number of samples correctly classified by the i^{th} classifier but are not correctly classified by the j^{th} classifier. N_{00} and N_{01} is inversion of N_{11} and N_{10}, respectively.

The Q-statistic diversity of the i^{th} layer is computed by averaging the diversity of pairwise of selected classifiers. It is noted that the lower the value of the Q-statistic, the higher the value of the ensemble diversity. The second objective, which aims to maximize the ensemble diversity, is given by:

$$\min_{E^{(i)}} \left\{ \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} Q_{ij} \right\} \tag{10}$$

in which m is the number of selected classifiers.

3.3 Algorithms

In the training process, MULES receives the inputs including the training data $\mathcal{D}$, the validation data $\mathcal{V}$, the learning algorithms $\mathcal{K}$ and early stopping rounds T_{stop}. In each layer, MULES searches for the optimal subset of classifiers and features using bi-objective optimization. Among the Evolutionary Algorithms introduced in solving the multi-objective optimization problems, the non-dominated sorting genetic algorithm II (NSGA-II) is one of the most popular and effective methods [26]. NSGA-II was designed with the elitism and diversity preserving characteristics so as to find the Pareto-optimal solution (a set of non-dominated solutions) [6]. In this study, we use NSGA-II in each layer to solve the combinational optimization problem given in (8) and (10) (line 4). Since there is no clear relationship between accuracy and diversity of an ensemble [12], the use of NSGA-II makes these objectives to be considered separately, thus maintaining the richness of both criteria in the evolution process. The final result is still the prediction accuracy as it is the main objective of interests [22, 26]. Therefore, we simply choose the chromosome with the best accuracy from the last generation of NSGA-II as the final selected individual.

Based on the encoding of the selected individual, we get the set of selected classifiers $\mathcal{H}^{(i)}$ and their associated features $\mathcal{F}^{(i)}$. The selected algorithms $\mathcal{K}^{(i)}$ and $\mathcal{F}^{(i)}$ is then used on $\mathcal{L}^{(i-1)}$ with the Cross-Validation procedure to generate the predictions for the training instances at the i^{th} layer i.e. a $N \times (MK)$ matrix $\mathcal{P}^{(i)}$. In line 8, $\mathcal{P}^{(i)}$ is concatenated with the original training data $\mathcal{L}^{(0)}$ to form the input training data $\mathcal{L}^{(i)}$ for the $(i+1)^{th}$ layer.

In this study, we evaluate the predictive performance of MULES on $\mathcal{V}$ on each layer to automatically determine the number of layers. In line 9, we used the selected $\mathcal{H}^{(i)}$ and the selected features for each classifier $\mathcal{F}^{(i)}$ on$\mathcal{V}_{i-1}$ to obtain the predictions $\mathcal{P}^{(i)}_{\mathcal{V}_i}$ for instances in the validation set at the i^{th} layer. As mentioned in (4), we apply Sum Rule on $\mathcal{P}^{(i)}_{\mathcal{V}_i}$ to obtain the predicted class label Y (line 10). By comparing Y and the ground truth of class labels of the instances in $\mathcal{V}$, we can calculate the classification error rate of the i^{th} layer on $\mathcal{V}$ (line 11). The predictions $\mathcal{P}^{(i)}_{\mathcal{V}_i}$ is also concatenated with the original validation set $\mathcal{V}$ to obtain $\mathcal{V}_{(i)}$ for the evaluation at the $(i+1)^{th}$ layer.

We use a checkpoint to save the current best result and the number of layers when MULES enhances its performance on the validation set (line 13-16). After a specific number of layers, if the classification error on the validation part does not improve, we stop growing new layers and then use the checkpoint to choose the optimal number of layers.

In the testing process, an unlabeled instance is pass through all layers in MULES until reaching the last layer. In the i^{th} layer, by referencing $\mathcal{F}^{(i)}$, we can choose the features of the test instance for each selected classifier. The predictions of the selected classifiers for the test instance are concatenated to the original features to form the new training data for the next layer. We use Sum Rule on the predictions of the selected classifiers at the last layer to assign a class label for the test instance.

Algorithm: MULES

Input:

- Training data $\mathcal{D} = [(\mathbf{x_n}, \hat{y}_n), n = 1...N]$
- Validation data $\mathcal{V} = [(\mathbf{x_i}, \hat{y}_i), i = 1...|\mathcal{V}|]$
- K learning algorithm $\mathcal{K}$
- Early stopping rounds T_{stop}

Output: List of selected classifiers and their associated features at layer i

Initizlize $\mathcal{L}^{(0)} = \mathcal{D}, \mathcal{V}_0 = \mathcal{V}, i = 0, best\_error\_model = 1.0$;

while *True* **do**
  i++ //Train layer i ;
  Apply NGSA-II with (8) and (10) to obtain optimal encoding E for layer (i) ;
  Get the set of selected classifier $\mathcal{H}^{(i)}$ and their associated features $\mathcal{F}^{(i)}$ from E ;
  Get the selected algorithm $\mathcal{K}^{(i)}$;
  $\mathcal{P}^{(i)} = Cross\_Validation(\mathcal{K}^{(i)}, \mathcal{F}^{(i)}, \mathcal{L}^{(i-1)})$;
  $\mathcal{L}^{(i)} = \mathcal{P}^{(i)} \cup \mathcal{L}^{(0)}$;
  $\mathcal{P}^{(i)}_{\mathcal{V}_i} = predict(\mathcal{H}^{(i)}, \mathcal{F}^{(i)}, \mathcal{V}_{(i-1)})$;
  $Y = Sum\_Rule\_predict(\mathcal{P}^{(i)}_{\mathcal{V}_i})$ by (4) ;
  $error = Loss\_function(Y, \mathcal{V})$;
  $\mathcal{V}^{(i)} = \mathcal{P}^{(i)}_{\mathcal{V}_i} \cup \mathcal{V}_0$;
  if $error < best\_error\_model$ **then**
    $best\_error\_model = error$;
    $T_{layer} = i$;
  end
  if *error does not decrease after* T_{stop} *layers* **then**
    Break
  end
end
Return $\left[\mathcal{H}^{(i)}\mathbf{and}\mathcal{F}^{(i)}\right], i = 1, ..., T_{layers}$;

4 EXPERIMENTAL STUDIES

4.1 Configurations

MULES was constructed with five learning algorithms: K Nearest Neighbor (KNN, K was set to 5), Logistic Regression, Naïve Bayes (Bernoulli distribution was used), Random Forest (with 200 estimators), and Decision Tree. All these methods were implemented from the scikit-learn library with default parameters. We followed the experiments in [30] in which 80% of labeled data is used for the training part and the remainder is used for the validation part. We used the 2-fold Cross-Validation in one layer to generate the predictions for the training part. The layer growing process is stopped if the classification error rate on the validation part does not improve after 5 layers. For NSGA-II algorithm using in each layer, the maximum number of generations was set to 100 and the population size was set to 50.

We used some well-known benchmark algorithms to evaluate the performance of MULES: Random Forest (with 2000 trees), gcForest (4 forests and 500 trees in each forest), XgBoost (with 2000 trees), and Multi-Layer Perceptron (MLP). As the performance of MLP significantly depends on the network structure, we performed grid search on different parameters and reported the best result for the comparison. For MLP, we followed the experiments in [30] in which it was constructed with different configurations: input-30-20-output, input-50-30-output, and input-70-50-output.

We used the Friedman test to test the null hypothesis that all methods perform equally on all datasets. If the *P-Value* of this test is smaller than a significant threshold e.g. 0.05, we reject the null hypothesis and conduct the Nemenyi post-hoc test for pairwise comparison on all datasets [7]. The experiments were conducted on 33 datasets selected from various sources such as the UCI Machine Learning Repository and OpenML [1].

4.2 Experimental Results

Comparison to baselines: The prediction error rates of MULES and five benchmark algorithms are presented in Table 1. Some observations can be made:

- Based on the Friedman test, the null hypothesis was rejected with the P-Value. The Nemenyi test shows that MULES is better than XgBoost, gcForest, and MLP.
- MULES achieves the lowest average rank among all methods (rank value 1.76). On the 33 datasets, MULES ranks first in 16 cases (48.5%) and ranks second in 13 cases (39.4%). Although MULES performs poorly on 4 datasets, i.e. it ranks fourth on the Breast-Cancer and Twonorm datasets and ranks third on the Cleveland and Spambase datasets, the prediction error rates of MULES and the first rank method are not significant differences (0.0488 vs. 0.0244 of gcForest on Breast-Cancer dataset, for example).
- Random Forest and XgBoost rank second and third with average rank value 2.59 and 3.12, respectively. Random Forest ranks first on 9 datasets while XgBoost ranks first on 4 datasets. Random Forest is better than MULES on only two datasets Hayes-Roth (0.1250 vs. 0.1667) and Wine_white (0.3014 vs. 0.3449). In contrast, MULES is better than Random Forest on 6 datasets namely Chess-krvk (about 6% better), Embryonal (more than 10% better), Hill-valley (about 30% better), Madelon (about 10% better), Tic-tac-toe and Vehicle (about 5% better).
- gcForest is worse than MULES in our experiment. One some datasets such as Chess-krvk, Electricity, Hill-valley, and Isolet, gcForest performs poorly and by far worse than MULES. gcForest ranks first on three datasets namely Breast-Cancer, Marketing, and Cleveland, but the differences in comparison to classification results of MULES are not significant.
- MLP is the poorest method in our experiment. Although MLP was run with different configurations and we reported the best result for comparisons, its performance is by far worse than MULES. On some datasets like Ring, Satimage, and

[1]The experimental datasets are Biodeg, Breast-Cancer, Breast-Tissue, Chess-krvk, Cleveland, Contraceptive, Electricity, Embryonal, Hayes-roth, Hill-valley, Isolet, Letter, Leukemia, Madelon, Magic, Marketing, Musk1, Phoneme, Ring, Satimage, Skin-NonSkin, Spambase, Texture, Tic-tac-toe, Titanic, Twonorm, Vehicle, Vertebral, Waveform_w_noise, Waveform_wo_noise, Wine, Wine_red, Wine_white. The detail of these datasets can be found in the Supplement Material

Table 1: Classification error rate and ranking of the benchmark algorithms and MULES

	gcForest	MLP	Random Forest	XgBoost	MULES
Biodeg	0.1325 (4)	**0.1073 (1)**	0.1230 (2.5)	0.1546 (5)	0.1230 (2.5)
Breast-Cancer	**0.0244 (1)**	0.0488 (4)	0.0439 (2)	0.0488 (4)	0.0488 (4)
Breast-Tissue	0.3125 (3.5)	0.3750 (5)	**0.2813 (1.5)**	0.3125 (3.5)	**0.2813 (1.5)**
Chess-krvk	0.1882 (3)	0.3571 (5)	0.1784 (2)	0.2526 (4)	**0.1280 (1)**
Cleveland	**0.4000 (1.5)**	0.5000 (5)	**0.4000 (1.5)**	0.4444 (4)	0.4111 (3)
Contraceptive	0.4344 (3)	**0.4027 (1)**	0.4525 (4)	0.4615 (5)	0.4276 (2)
Electricity	0.1989 (5)	0.1950 (4)	0.0937 (3)	0.0926 (2)	**0.0674 (1)**
Embryonal	0.5000 (3.5)	**0.3889 (1.5)**	0.5556 (5)	0.5000 (3.5)	**0.3889 (1.5)**
Hayes-roth	0.1667 (2.5)	0.2292 (5)	**0.1250 (1)**	0.1875 (4)	0.1667 (2.5)
Hill-valley	0.4148 (5)	0.1058 (2)	0.3407 (4)	0.3214 (3)	**0.0234 (1)**
Isolet	0.1863 (5)	0.0504 (2)	0.0603 (4)	0.0543 (3)	**0.0483 (1)**
Letter	0.0360 (2)	0.0658 (5)	0.0427 (4)	0.0373 (3)	**0.0332 (1)**
Leukemia	0.0909 (4)	0.0909 (4)	**0.0455 (1.5)**	0.0909 (4)	**0.0455 (1.5)**
Madelon	0.3550 (4)	0.4550 (5)	0.3300 (3)	0.3117 (2)	**0.2283 (1)**
Magic	0.1539 (4)	0.1604 (5)	**0.1193 (1)**	0.1213 (3)	0.1197 (2)
Marketing	**0.6418 (1)**	0.6597 (3)	0.6709 (4)	0.6733 (5)	0.6563 (2)
Musk1	0.1538 (2.5)	0.1748 (5)	0.1538 (2.5)	0.1678 (4)	**0.1329 (1)**
Phoneme	0.1726 (5)	0.1369 (4)	**0.0943 (1)**	0.1307 (3)	0.0986 (2)
Ring	0.0351 (3)	0.1568 (5)	0.0410 (4)	**0.0284 (1)**	0.0342 (2)
Satimage	0.1253 (4)	0.1636 (5)	0.0777 (3)	**0.0284 (1)**	0.0756 (2)
Skin-NonSkin	1.4350E-02 (5)	8.5693E-04 (4)	5.5769E-04 (3)	4.7607E-04 (2)	**4.3500E-04 (1)**
Spambase	0.0608 (4)	0.0644 (5)	**0.0449 (1)**	0.0478 (2)	0.0492 (3)
Texture	0.1564 (5)	**0.0067 (1)**	0.0248 (4)	0.0152 (3)	0.0145 (2)
Tic-tac-toe	0.1632 (5)	0.0729 (3)	0.0764 (4)	**0.0000 (1)**	0.0208 (2)
Titanic	0.2466 (2)	0.2496 (4)	0.2496 (4)	0.2496 (4)	**0.2284 (1)**
Twonorm	0.0306 (5)	0.0293 (3.5)	0.0243 (2)	**0.0239 (1)**	0.0293 (3.5)
Vehicle	0.2717 (4)	0.3110 (5)	0.2362 (3)	0.2283 (2)	**0.1811 (1)**
Vertebral	0.2043 (4.5)	0.2043 (4.5)	**0.1505 (1)**	0.1935 (3)	0.1720 (2)
Waveform_w_noise	0.1433 (4)	0.1633 (5)	0.1333 (2)	0.1380 (3)	**0.1233 (1)**
Waveform_wo_noise	0.1567 (3.5)	0.1567 (3.5)	0.1427 (2)	0.1713 (5)	**0.1353 (1)**
Wine	**0.0000 (2)**	0.6111 (5)	**0.0000 (2)**	0.0556 (4)	**0.0000 (2)**
Wine_red	0.4146 (4)	0.4354 (5)	0.3354 (2)	0.3604 (3)	**0.3250 (1)**
Wine_white	0.4252 (4)	0.4837 (5)	**0.3014 (1)**	0.3490 (3)	0.3449 (2)
Average Ranking	3.59	3.94	2.59	3.12	**1.76**

Table 2: An example of the obtained configuration for MULES on Tic-tac-toe dataset

	Classifiers	Features	Encoding
Layer 1	Random Forest (200)	6 original features	011010111
	Decision Tree	6 original features	111110100
	KNN(5)	5 original features	100100111
Layer 2	Random Forest (200)	5 original features + 1 prediction	100010111 | 000100
	Naïve Bayes	7 original features + 5 predictions	001111111 | 111110
	KNN(5)	4 original features + 4 predictions	100101100 | 111010
	Logistic Regression	6 original features + 2 predictions	011110110 | 010010
Layer 3	KNN(5)	4 original features + 8 predictions	101001100 | 10111111
	Logistic Regression	3 original features + 6 predictions	011000100 | 10011111
Layer 4	KNN(5)	3 original features + 2 predictions	001001001 | 1010
	Logistic Regression	3 original features + 1 prediction	100010001 | 0010

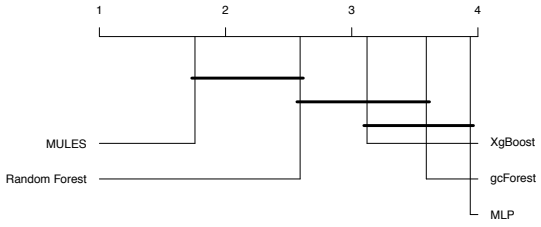


Figure 3: The Nememyi test result

Vehicle, the classification error rates of MLP are significantly higher than those of MULES.

The number of layers and configuration: Figure 4 presents the comparisons between the number of layers generated by gcForest and MULES on the experimental datasets. Like gcForest, MULES can automatically determine the number of layers based on its prediction performance on the validation set. On average, MULES generated 4 layers of ensemble of classifiers. This means going deeply can improve the performance of the deep model on the validation data of some datasets. Moreover, MULES generated more number of layers than gcForest, 4 compared to 3.7 on average. Exceptionally, both methods generate only one layer on 9 and 6 datasets. In these cases, going deeply with the new proposed input data does not bring benefits to the ensemble system.

We further analyze the benefits of the multi-layer architecture in MULES. Figure 5 shows the reductions of the prediction error rates on the test data of MULES through layers in the deep model. On Hayes-Roth dataset, for example, where MULES generates the model with 2 layers, the prediction error rate reduces from 0.1875 to 0.1667. On Tic-Tac-Toe dataset, the prediction error rate reduces from 0.1042 to 0.059 from the first layer to second layer, and continue to reduce to 0.0486 at the third layer and to 0.0208 at the fourth layer. This figure demonstrates the advantages of layer-by-layer

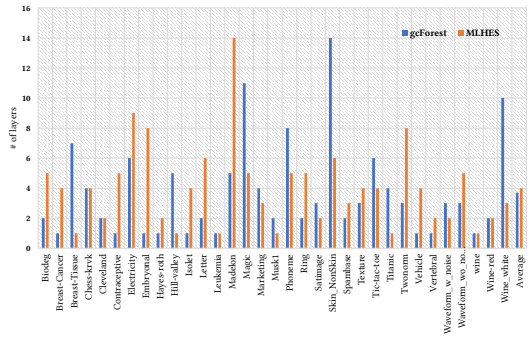


Figure 4: The number of layers obtained by gcForest and MULES.

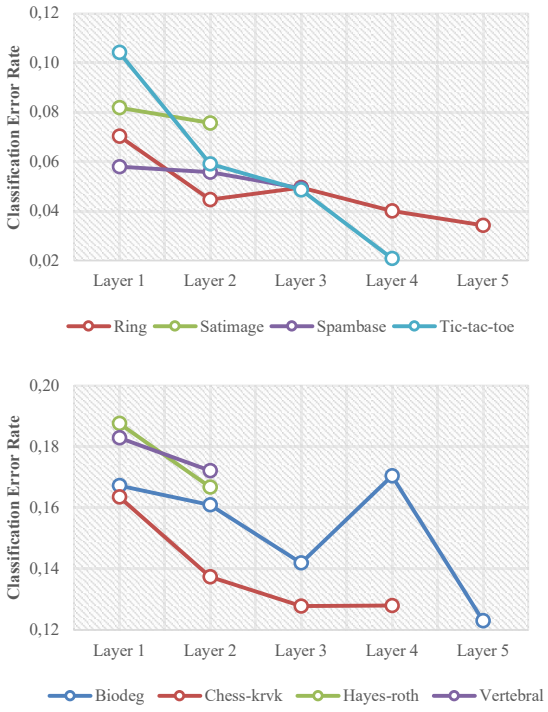


Figure 5: The changes of classification error rate in each layer.

processing in MULES since the classification system gets better results with deeper layer.

We illustrate an example of the obtained configuration for MULES on the Tic-tac-toe dataset. At each layer, MULES selected the suitable classifiers and their features to generate the input for the next layer. In detail, on the first layer, MULES chose 3 classifiers: Random Forest, Decision Tree, and KNN. Each classifier used its own features selected from the original features. In the second layer, classifiers except Decision Tree were selected with different feature sets obtained from the original features and predictions. On the third and fourth layers, two classifiers KNN and Logistic Regression were selected with different feature sets. By selecting the suitable classifiers in each layer and suitable features for each classifier, MULES can obtain high prediction accuracy and effectiveness in resource usage in terms of memory and computation requirements.

Classification time: Although MULES takes much higher training time than gcForest, the classification time of MULES is lower than gcForest. On Tic-tac-toe dataset, for example, MULES used 3154.86 second for training process compared to only 311.78 of gcForest. Meanwhile, gcForest used 0.62 second to classify all test instances while MULES only used 0.26 second.

MULES obtains a subset of classifiers and their features in each layer. From the obtained configuration for Tic-tac-toe dataset in Table 2, after the evolution process, only 3 and 4 classifiers were kept in the first and second layer. The third and fourth layer only have 2 classifiers. Therefore, only 11 classifiers are maintained in MULES to classify instances on this dataset. That makes MULES takes less time during classification. In contrast, gcForest used 4 Random Forests involving 500 trees in each forest. MLP also takes high computation for the training process as its configuration is somehow problem-dependent that requires a procedure to search for the optimal one. It is noted that like population-based Evolution Algorithms, NSGA-II can be implemented in parallel. This can further reduce the training time of MULES.

5 CONCLUSIONS

In summary, we introduced a Multi-Layer Heterogeneous Ensemble System (MULES) inspired by the layer-by-layer processing of DNNs. MULES includes several layers of the ensemble of different classifiers in which the classifiers in one layer train on the new training data generated by the preceding layer. The new training data for one layer is the concatenation of the predictions of the classifiers in the preceding layer and the original training data. We train a combining algorithm on the predictions of classifiers in the last layer for the final collaborated prediction. Since the ensemble in each layer can contain the unnecessary classifiers which increase the prediction error of the ensemble, we propose an Evolutionary Algorithm-based method to select the optimal set of classifiers and their features on each layer. The optimization problem is considered under two objectives concerning the prediction error and ensemble diversity. We used NSGA-II, a popular and effective multi-objective evolutionary algorithm, to solve this optimization problem.

Experiments on 33 datasets confirm that MULES is better than MLP, Random Forest, gcForest, and XgBoost in terms of predictive performance and efficiency.

Two solutions to enhance the performance of MULES could be implemented in the future. First, in this study we concatenated the original training data and the predictions of classifiers in one layer to generate the input data for the next layer. In general, MULES can generate multiple layers and its prediction accuracy becomes better on the deeper layers. The exception occurred on 6 datasets where going deeply does not achieve any improvements for the classification process. A new input data is needed to populate the deep model in these cases. Second, we used NSGA-II to search for the set of classifiers and their features in each layers. Parallel implementation of NSGA-II can be used in MULES to reduce its training time.

REFERENCES

[1] Marija Bacauskiene, Antanas Verikas, Adas Gelzinis, and Donatas Valincius. 2009. A feature selection technique for generation of classification committees and its application to categorization of laryngeal images. *Pattern Recognition* 42, 5 (2009), 645–654.

[2] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[3] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. 2018. Autostacker: A compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 402–409.

[4] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[5] Yijun Chen and Man Leung Wong. 2011. Optimizing stacking ensemble by an ant colony optimization approach. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. 7–8.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[7] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7, Jan (2006), 1–30.

[8] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research* 15, 1 (2014), 3133–3181.

[9] Kyung-Joong Kim and Sung-Bae Cho. 2008. An evolutionary algorithm approach to optimal ensemble classifiers for DNA microarray data analysis. *IEEE Transactions on Evolutionary Computation* 12, 3 (2008), 377–388.

[10] Josef Kittler, Mohamad Hatef, Robert PW Duin, and Jiri Matas. 1998. On combining classifiers. *IEEE transactions on pattern analysis and machine intelligence* 20, 3 (1998), 226–239.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[12] Ludmila I Kuncheva and Christopher J Whitaker. 2003. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning* 51, 2 (2003), 181–207.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[14] Pablo Ribalta Lorenzo and Jakub Nalepa. 2018. Memetic evolution of deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 505–512.

[15] Tien Thanh Nguyen, Alan Wee-Chung Liew, Xuan Cuong Pham, and Mai Phuong Nguyen. 2014. Optimization of ensemble classifier system based on multiple objectives genetic algorithm. In *2014 International Conference on Machine Learning and Cybernetics*, Vol. 1. IEEE, 46–51.

[16] Tien Thanh Nguyen, Alan Wee-Chung Liew, Minh Toan Tran, and Mai Phuong Nguyen. 2014. Combining multi classifiers based on a genetic algorithm–a gaussian mixture model framework. In *International Conference on Intelligent Computing*. Springer, 56–67.

[17] Tien Thanh Nguyen, Alan Wee-Chung Liew, Minh Toan Tran, Thi Thu Thuy Nguyen, and Mai Phuong Nguyen. 2014. Fusion of classifiers based on a novel 2-stage model. In *International Conference on Machine Learning and Cybernetics*. Springer, 60–68.

[18] Tien Thanh Nguyen, Anh Vu Luong, Thi Minh Van Nguyen, Trong Sy Ha, Alan Wee-Chung Liew, and John McCall. 2019. Simultaneous meta-data and meta-classifier selection in multiple classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 39–46.

[19] Tien Thanh Nguyen, Mai Phuong Nguyen, Xuan Cuong Pham, Alan Wee-Chung Liew, and Witold Pedrycz. 2018. Combining heterogeneous classifiers via granular prototypes. *Applied Soft Computing* 73 (2018), 795–815.

[20] Ioannis Partalas, Grigorios Tsoumakas, and Ioannis Vlahavas. 2010. An ensemble uncertainty aware measure for directed hill climbing ensemble pruning. *Machine Learning* 81, 3 (2010), 257–282.

[21] Zhiquan Qi, Bo Wang, Yingjie Tian, and Peng Zhang. 2016. When ensemble learning meets deep learning: a new deep support vector machine for classification. *Knowledge-Based Systems* 107 (2016), 54–60.

[22] Jacob Schrum. 2018. Evolving indirectly encoded convolutional neural networks to play tetris with low-level features. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 205–212.

[23] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. 2019. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation* (2019).

[24] Lev V Utkin, Maxim S Kovalev, and Anna A Meldo. 2019. A deep forest classifier with weights of class probability distribution subsets. *Knowledge-Based Systems* 173 (2019), 15–27.

[25] Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, Vol. 1. IEEE, I–I.

[26] Yuyan Wang, Dujuan Wang, Na Geng, Yanzhang Wang, Yunqiang Yin, and Yaochu Jin. 2019. Stacking-based ensemble learning of decision trees for interpretable prostate cancer detection. *Applied Soft Computing* 77 (2019), 188–204.

[27] Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*. 1379–1388.

[28] Bing Xue, Mengjie Zhang, Will N Browne, and Xin Yao. 2015. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation* 20, 4 (2015), 606–626.

[29] Zhiwen Yu, Daxing Wang, Jane You, Hau-San Wong, Si Wu, Jun Zhang, and Guoqiang Han. 2016. Progressive subspace ensemble learning. *Pattern Recognition* 60 (2016), 692–705.

[30] Zhi-Hua Zhou and Ji Feng. 2017. Deep Forest: Towards An Alternative to Deep Neural Networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 3553–3559. https://doi.org/10.24963/ijcai.2017/497