HW-Forest: Deep Forest with Hashing Screening and Window Screening

PENGFEI MA, School of Artificial Intelligence, Hebei University of Technology, China
YOUXI WU, School of Artificial Intelligence, Hebei University of Technology, China and Hebei Key Laboratory of Big Data Computing, China
YAN LI, School of Economics and Management, Hebei University of Technology, China
LEI GUO, State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, China
HE JIANG, School of Software, Dalian University of Technology, China
XINGQUAN ZHU, Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, USA
XINDONG WU, Key Laboratory of Knowledge Engineering with Big Data (the Ministry of Education of China), Hefei University of Technology, China and Mininglamp Academy of Sciences, Mininglamp Technology, China

As a novel deep learning model, gcForest has been widely used in various applications. However, current multi-grained scanning of gcForest produces many redundant feature vectors, and this increases the time cost of the model. To screen out redundant feature vectors, we introduce a hashing screening mechanism for multi-grained scanning and propose a model called HW-Forest which adopts two strategies: hashing screening and window screening. HW-Forest employs perceptual hashing algorithm to calculate the similarity between feature vectors in hashing screening strategy, which is used to remove the redundant feature vectors produced by multi-grained scanning and can significantly decrease the time cost and memory consumption. Furthermore, we adopt a self-adaptive instance screening strategy called window screening to improve the performance of our approach, which can achieve higher accuracy without hyperparameter tuning on different datasets. Our experimental results show that HW-Forest has higher accuracy than other models, and the time cost is also reduced.

This work was supported by National Natural Science Foundation of China (61976240, 52077056, 91746209), National Key Research and Development Program of China (2016YFB1000901), Natural Science Foundation of Hebei Province, China (Nos. F202020203, E2020202033), and Graduate Student Innovation Program of Hebei Province (CXTZZS2021026).

Authors’ addresses: P. Ma, School of Artificial Intelligence, Hebei University of Technology, Tianjin 300401, China; email: mpfei1995@163.com; Y. Wu (corresponding author), School of Artificial Intelligence, Hebei University of Technology, Tianjin 300401, China; email: wuc567@163.com; Y. Li, School of Economics and Management, Hebei University of Technology, Tianjin 300401, China; email: lywuc@163.com; L. Guo, State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, Tianjin 300401, China; email: guoshengrui@163.com; H. Jiang, School of Software, Dalian University of Technology, Dalian 116023, China; email: jianghe@dlut.edu.cn; X. Zhu, Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, FL 33431; email: xzhu3@fau.edu; X. Wu, Key Laboratory of Knowledge Engineering with Big Data (The Ministry of Education of China), Hefei University of Technology, Hefei 230009, China, and Mininglamp Academy of Sciences, Mininglamp Technology, Beijing 100084, China; email: xwu@hfut.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.
1556-4681/2022/07-ART123 $15.00
https://doi.org/10.1145/3532193
1 INTRODUCTION

Deep learning [1] can obtain better performance than classical machine learning methods [2, 3] and has been widely applied in various fields, such as image processing [4], speech recognition [5], and text processing [6]. However, deep learning models based on neural network involve large numbers of hyperparameters and depend on a parameter tuning process. More importantly, theoretical analyses of neural network models are extremely difficult [7, 8]. There are therefore several reasons for exploring the development of a novel deep learning model. Based on the recognition that layer-by-layer processing and in-model feature transformation are key aspects of neural network models, Zhou and Feng [9] proposed a novel deep learning model called gcForest.

Inspired by neural network models, gcForest adopts a cascade structure to process data, and contains many random forests at each level of the cascade structure. Multi-grained scanning is employed, in which, as an in-model feature transformation mechanism, a sliding window is used to scan high-dimensional raw data. The main advantage of gcForest is that it does not need backpropagation at the training stage, which makes it much easier to train than the neural network model. Recently, gcForest has achieved excellent performance on a broad range of tasks, such as cancer detection [10] and remote sensing [11]. However, although existing experimental results show that larger models may provide higher accuracy, gcForest is difficult to implement with large models. The reason for this is that multi-grained scanning transforms a raw instance of high-dimension data into hundreds or thousands of novel image instances, and inputs them into all levels of the cascade structure. Thus, gcForest imposes high time costs and memory consumptions to realize learning.

To reduce the time costs and memory requirements of gcForest [9], Pang et al. [12] proposed gcForestcs, in which a confidence screening mechanism is introduced into the general framework of gcForest. Confidence screening mechanism selects instances according to their confidences, which are produced by their prediction vectors. More specifically, all instances are divided into two subsets: high-confidence and low-confidence. Instances in the high-confidence set are used to produce a final prediction at the current level, while the instances in the low-confidence set are passed to the next level. The key aspect of gcForestcs is confidence screening, and its threshold is produced by a simple and effective rule. However, this rule cannot avoid the mispartitioning of instances, which means that some low-accuracy instances are classified into the high-confidence set. To tackle this issue, we previously proposed DBC-Forest [13], in which binning confidence screening is used to replace confidence screening. More concretely, DBC-Forest packs all instances into bins based on their confidence values, and adopts the instance accuracy of the bins to avoid mispartitioning of instances. Experimental results showed that DBC-Forest achieved better performance than gcForestcs.

However, current deep forest models have two issues. First, multi-grained scanning produces some redundant feature vectors which increase time costs. The reason for producing them is that
some instances have the same features. Second, in cascade structure, the thresholds of these confidence screening mechanisms depend on tuning hyperparameters. In other words, they cannot achieve the best performance when the hyperparameters are not suitable for some datasets. To improve the performance of the deep forest models, we propose HW-Forest, which adopts two screening mechanisms: hashing screening and window screening. Hashing screening reduces the number of redundant feature vectors based on their perceptual hashing values, and can also be applied to other deep forest models, since the mechanism is an improvement on gcForest [9]. Furthermore, inspired by the binning confidence screening process used in DBC-Forest [13], a self-adaptive mechanism called window screening is proposed, in which a window is adopted to screen instances. Unlike DBC-Forest [13], window screening produces thresholds automatically for different datasets without requiring hyperparameter tuning.

The contributions of this article can be summarized as follows.

1. To improve the performance and efficiency of the deep forest algorithm, we propose a novel deep forest model called HW-Forest which adopts two screening mechanisms: hashing screening and window screening.
2. In HW-Forest, hashing screening is used to remove the redundant feature vectors produced by multi-grained scanning, which significantly decreases time costs and memory consumptions.
3. A self-adaptive confidence screening is used to realize window screening, which can achieve higher accuracy without hyperparameter tuning on different datasets.
4. Our experimental results show that the proposed hashing screening mechanism can not only reduce the running time of HW-Forest, but also reduce the running time of other deep forest methods. More importantly, HW-Forest yields better performance than other state-of-the-art approaches.

The remainder of this article is organized as follows. Section 2 introduces related work on the deep forest algorithm. Section 3 describes perceptual hashing and DBC-Forest. Section 4 introduces HW-Forest and its two screening mechanisms. Section 5 presents experimental results. Finally, conclusions are drawn in Section 6.

2 RELATED WORK

As a novel deep learning model, deep forest gives higher accuracy than deep neural networks, and it has therefore been applied in many fields, such as disease diagnosis, electricity consumption monitoring, emotion recognition, rice seeds selection, topic classification for tweets, and image retrieval. In disease diagnosis, Sun et al. [14] proposed AFS-DF, in which a feature selection mechanism was adopted to classify cases of COVID-19. To improve the accuracy of classification of cancer subtypes, Guo et al. [15] and Dong et al. [16] proposed the boosting cascade deep forest and multi-weighted gcForest algorithms, respectively. In electricity consumption monitoring, Wang et al. [17] proposed a novel voltage-current trajectory classification model based on deep forest. In emotion recognition, Cheng et al. [18] applied deep forest to emotion recognition based on electroencephalography. Fang et al. [19] proposed a multi-feature deep forest approach in which a multi-feature structure was employed to extract power spectral density and differential entropy features. In rice seeds selection, Zhang et al. [20] combined hyperspectral imaging technology with deep forest to identify frost-damaged rice seeds. In tweet topic classification, a semantic deep forest technique was developed [21], in which a semantic layer was introduced into the cascade structure of deep forest. In image retrieval, Zhou et al. [22] proposed a deep forest hashing approach to learn shorter binary codes. In economics, Ma et al. [23] proposed a cost-sensitive deep forest algorithm that assigned a specific cost for each misclassification, to improve the accuracy of deep forest in the
area of price prediction. In addition, some researchers analyzed the generalization performance of deep forest. For example, Lyu et al. [24] proposed an additive casForest model which offers a novel understanding of margin theory and representation learning. Arnould et al. [25] investigated deep forest underlying mechanisms, and the results show that the shallow tree network can enhance the performance of decision trees. To improve the performance of deep forest, Chen et al. [26] proposed the interaction-based feature concatenation for deep forests, which has greater diversity, and the results report that the order of learned interactions can be increased layer by layer.

However, the performance of deep forest needs to be further improved, since it is inefficient on datasets with larger numbers of instances. The reason is that gcForest adopts a cascade forest structure to obtain the final prediction [9]. Each level of cascade receives feature information processed by its preceding level, and outputs its processing result to the next level. Therefore, all instances reach the final level. This process has high time costs and memory requirements. To overcome this shortage, an approach called gcForest was proposed [12], in which the instances with high-confidence are used to produce a final prediction at the current level, while the instances with low-confidence are passed to the next level. Inspired by gcForestcs [12], Utkin et al. [27] proposed an adaptive weighted deep forest which can adaptively weight each training instance at each cascade level. Moreover, Yang et al. [28] proposed a multi-label learning with deep forest model which employed a measure-aware feature reuse mechanism to retain valuable presentation in the previous layer, and a measure-aware layer growth was adopted to limit the complexity of the model with different performance measures.

Although gcForestcs [12] has better performance in terms of training time than the original model on many datasets, especially in image processing applications, gcForestcs has poorer prediction accuracy than the original model on many datasets, since some low-accuracy instances in gcForestcs model are classified into the high-confidence set. This phenomenon is called mispartitioning of instances. To avoid the mispartitioning, DBC-Forest was proposed, which employed a binning confidence mechanism [13]. We will introduce the principle of DBC-Forest in Section 3.2. DBC-Forest [13] can avoid the mispartitioning of instances and thus improves the performance. However, the performance of DBC-Forest depends on the hyperparameter $k$, which represents the number of bins used to partition instances. To avoid using hyperparameter $k$, we will propose a window screening mechanism which adopts a self-adaptive confidence screening approach to achieve higher accuracy without hyperparameter tuning on different datasets.

In summary, the models described above pay less attention to the redundant feature vectors, which not only increase the time cost but also decrease the accuracy of the model. To further improve the performance of deep forest, we propose a novel model called HW-Forest, in which hashing screening and window screening are applied. Our experimental results show that HW-Forest achieves better performance than other state-of-the-art models.

### 3 PRELIMINARIES

Section 3.1 introduces perceptual hashing algorithms. Section 3.2 reviews the principles underlying DBC-Forest, including the use of binning confidence screening.

#### 3.1 Perceptual Hashing

Perceptual hashing [29] is an effective approach for image retrieval in which images are converted into binary sequences for retrieval. The most representative perceptual hashing algorithms are aHash [30], pHash [31], and dHash [32]. To process data efficiently, we employ aHash to screen out redundant feature vectors, using a hashing screening mechanism. The main steps of aHash are as follows:

**Step 1:** Resize the image (this is a fast way to match an image of any size).
Step 2: Convert the resized image to grayscale.
Step 3: Compute the mean value of the grayed image pixels.
Step 4: Compare each pixel value with the mean value, and set any pixels larger than the mean to one, and the other pixels to zero.
Step 5: Combine the binarization results into binary sequences.

Finally, aHash retrieves images by comparing their binary sequences. The process used by the aHash algorithm is illustrated in Figure 1.

3.2 DBC-Forest

As an improved version of gcForestcs, DBC-Forest adopts a binning confidence screening mechanism to avoid the mispartitioning of instances with low accuracy into high-confidence regions. More specifically, the binning confidence screening mechanism packs all instances into bins, and DBC-Forest then calculates the accuracy of all bins to locate mispartitioned instances. This mechanism can be summarized as follows.

Step 1: Calculate the confidence of each instance at the current level. We utilize confidence as the criterion, for instance, selecting which is generated by the estimated class vector. For example, if the estimated class vector, for instance, \( \mathbf{x} \) is (0.5, 0.3, 0.2), its confidence is 0.5.

Step 2: Rank instances based on their confidences. We then obtain instances \( (\mathbf{x}_{s_1}, \mathbf{x}_{s_2}, \ldots, \mathbf{x}_{s_m}) \), where \( \mathbf{x}_{s_1} \) is the highest confidence instance, and put these instances into \( k \) bins. Each bin \( b_t \) (\( 1 \leq t \leq k \)) can be obtained according to Equation (1):

\[
(\mathbf{x}_{s_{(m \times t - 1) + 1}}, \mathbf{x}_{s_{(m \times t - 1) + 2}}, \ldots, \mathbf{x}_{s_{(m \times t)}}) \in b_t.
\]

Step 3: Calculate the accuracy for each bin. The accuracy \( P_t \) of \( b_t \) is calculated according to Equation (2):

\[
P_t = \frac{\sum_{i=(m \times (t-1) + 1)}^{m \times t} 1[p(\mathbf{x}_{s_i}) = y_{s_i}]}{m/k},
\]

where \( p(\mathbf{x}_{s_i}) \) is the prediction of \( \mathbf{x}_{s_i} \) at the current level, and \( y_{s_i} \) is the label of \( \mathbf{x}_{s_i} \).

Step 4: Determine the threshold. In DBC-Forest, the accuracy of each bin is compared with the target accuracy \( T_A \) from 1 to \( k \). If \( P_{j+1} \) is the first bin for which the accuracy is lower than \( T_A \), then we utilize the confidence of \( \mathbf{x}_{s_{(m \times j+1)}} \) as the threshold.

Step 5: Compare the confidences of the instances with the threshold. If the confidence of the instance is larger than the threshold, then the instance is used to produce a final prediction at the current level; otherwise, the instance is passed to the next level.

4 PROPOSED METHOD

Section 4.1 introduces the hashing screening strategy. Section 4.2 presents the window screening strategy. Section 4.3 describes HW-Forest and illustrates its framework.
Fig. 2. The production of redundant feature vectors. We select two instances from the MNIST dataset, and assume the size of the slide window is $2 \times 2$.

Fig. 3. The relationships among $(x_1, x_2, \ldots, x_r)$, $(v_1, v_2, \ldots, v_r)$, and $(f_{1,1}, f_{1,2}, \ldots, f_{r,m})$. Feature vectors $f_{1,1}$ and $f_{1,2}$ are in group $v_1$, since they are the first feature vectors of different instances.

4.1 Hashing Screening

Multi-grain scanning is a structure for processing high-dimensional data that can achieve excellent performance on image data. However, the time cost of multi-grained scanning is increased by the presence of redundant feature vectors, which are produced by sampling the same features from instances. Figure 2 shows the production of a redundant feature vector by a sliding window.

To reduce the number of redundant feature vectors, we apply the aHash algorithm to evaluate the similarity between feature vectors and eliminate the high similarity feature vectors. This process can be summarized as follows.

Step 1: Group feature vectors according to their locations, which gives $(v_1, v_2, \ldots, v_r)$ according to Equation (3):

$$ (f_{r,1}, f_{r,2}, \ldots, f_{r,m}) \in v_r. \tag{3} $$

The relationships between these vectors are illustrated in Figure 3. For example, $f_{1,1}$ and $f_{1,2}$ are members of group $v_1$, and are the first feature vectors of instances $x_1$ and $x_2$, respectively.

Step 2: Obtain binary sequences for all feature vectors using aHash. For example, $(b_{r,1}(k_1), b_{r,1}(k_2), \ldots, b_{r,1}(k_c))$ is the binary sequence of $b_{r,1}$, where $c$ is the length of the binary sequence and $b_{r,1}(k_j)$ is obtained by aHash.

Step 3: Calculate the distances between the features in each group. In group $v_r$, the $j$th feature distance $v_r(k_j)$ can be calculated using Equation (4):

$$ v_r(k_j) = \begin{cases} \frac{\sum_{i=1}^{m} b_{r,i}(k_j)}{m}, & \frac{\sum_{i=1}^{m} b_{r,i}(k_j)}{m} < 0.5 \\ 1 - \frac{\sum_{i=1}^{m} b_{r,i}(k_j)}{m}, & \frac{\sum_{i=1}^{m} b_{r,i}(k_j)}{m} \geq 0.5 \end{cases} \tag{4} $$

Step 4: Calculate the distance between each pair of groups. This distance represents the differences in the feature vectors in the groups. The larger the distance, the greater the difference in the
Suppose we have four groups $\{v_1, v_2, \ldots, v_r\}$, for which the feature vectors are $(f_{1,1} = (0, 1), f_{1,2} = (0, 1), f_{2,1} = (1, 1), f_{2,2} = (1, 1), f_{3,1} = (1, 0), f_{3,2} = (0, 1), f_{4,1} = (0, 1), f_{4,2} = (1, 0))$. We know that the values of $d_{1,1}(k_1)$ and $d_{1,2}(k_1)$ are 0. Hence, according to Equation (4), the value of $v_1(k_1) = \frac{0+0}{2} = 0$. Similarly, the values of $(v_1(k_1), v_1(k_2)), (v_2(k_1), v_2(k_2))$, $(v_3(k_1), v_3(k_2))$, and $(v_4(k_1), v_4(k_2))$ are (0, 0), (0, 0), (0, 0.5), and (0.5, 0.5), respectively. Since $v_1(k_1) = 0.5$ and $v_1(k_2) = 0.5$, the distance of $v_1$ is 0.5, i.e., $d_4 = \frac{0.5+0.5}{2} = 0.5$. Thus, $(d_1, d_2, d_3, d_4)$ are (0, 0, 0.5, 0.5). We then rank the groups according to their distances. Finally, we obtain $(w_1, w_2, w_3, w_4)$, whose distances are $(e_1 = 0.5, e_2 = 0.5, e_3 = 0, e_4 = 0)$. Hence, $D = \sum_{i=1}^4 e_i = 0.5+0.5+0+0 = 1$. We now obtain the values of $(N(100), N(99)), \ldots, N(1))$. According to Equation (6), the values of $N(100), N(99), \ldots, N(51)$ are 2, since $e_1 + e_2 \geq 100 \times \frac{50}{100} = 1$, and $e_1 + e_2 \geq 51 \times \frac{50}{100} = 0.51$. Similarly, the values of $N(50), N(49), \ldots, N(1)$ are 1, since $e_1 \geq 50 \times \frac{50}{100} = 0.50$ and $e_1 \geq 1 \times \frac{50}{100} = 0.01$. We know that there are four groups, meaning that $r = 4$, and $\frac{p}{100} = \frac{1}{4} = 0.08$. According to Step 7, $p = 100$, since $N(100) - N(99)$ is 0, and this is the first value lower than 0.08. Since $p = 100$ and $N(100) = 2$, according to Equation (7), we select the second group distance $e_2$ as the threshold, i.e., $h = 2$. Finally, we eliminate $w_3$ and $w_4$, since their distances $e_3$ and $e_4$ are lower than $e_2$. $w_1$ and $w_2$ are converted to class vectors, and input into the cascade structure. Figure 4 illustrates the hashing screening mechanism.
Example 1 verifies that hashing screening can effectively eliminate the redundant feature vectors. To further introduce hashing screening mechanism, we analyze the time complexities of hashing screening and HW-Forest.

Suppose the number of instances is \( m \), and the multi-grained scanning window size is \( n \times n \). The time complexity consists of three parts: binary sequence producing, distance value calculating, and threshold producing. In the part of binary sequence producing, the time complexity is \( O(m \times n \times n \times a) \), where \( a \) is the number of feature vectors. In the part of distance value calculating, the time complexity is also \( O(m \times n \times n \times a) \). In the part of threshold producing, the time complexity is \( O(a) \). Therefore, the overall time complexity of hashing screening is \( O(m \times n \times n \times a + a) = O(m \times n \times n \times a) \).

We know that the time complexity of a random forest is \( O(M \times (u \times c \times \log(c))) \) [34], where \( M \) is the number of decision trees, \( u \) and \( c \) are the number of instances and the number of instance features of the model, respectively. For a deep forest model, \( c \) is \( n \times n \), since the window size is \( n \times n \). Moreover, if we do not use hashing screening strategy, the number of instances \( u \) is \( m \times a \), since there are \( m \) instances in the model and each instance has a feature vectors. Suppose a deep forest model uses \( K \) random forests. Therefore, the time complexity of training model without using the hashing screening strategy is \( O(K \times M \times (m \times n \times n \times a \times \log(n \times n))) \). Suppose the hashing screening strategy can find \( t \) redundant feature vector groups, which means that the number of instances drops to \( m \times (a - t) \). Hence, the time complexity of HW-Forest is \( O(K \times M \times (m \times n \times n \times (a - t) \times \log(n \times n)) + m \times n \times n \times a) \). Obviously, the greater the \( t \), the more training time will be reduced.

4.2 Window Screening

As an improved version of gcForestcs [12], DBC-Forest [13] avoids the mispartition of instances and thus improves the performance. However, its performance depends on the hyperparameter \( k \), which represents the number of bins used to partition instances. Specifically, the smaller the value of \( k \), the larger the threshold. Example 2 illustrates this phenomenon.

Example 2. Suppose we have 16 instances sorted by confidence, as illustrated in Figure 5. Of these, instances 1, 2, 4, 5, 6, 7, 8, and 11 are correct predictions, and the others are incorrect. We set two cases with \( k = 8 \) and \( k = 4 \) under the condition of hyperparameter \( TA = 70\% \). In the case of \( k = 8 \), instance 2 is selected as the threshold in DBC-Forest, since the accuracy of bin II is 50\%. In the case of \( k = 4 \), instance 8 is selected, since the accuracy of bin III is 25\%. This example shows that the performance of DBC-Forest depends on the hyperparameter \( k \). In other words, DBC-Forest cannot achieve high accuracy if hyperparameter \( k \) is not suitable for the dataset.

To address this drawback of DBC-Forest, we propose a window screening mechanism, in which a self-adaptive approach is used to calculate the threshold. Unlike the fixed window used in DBC-Forest, this approach uses windows with variable size. The mechanism can be summarized as follows.

Step 1: Rank the instances based on their confidences, and obtain \((x_1, x_2, \ldots, x_m)\).

Step 2: Establish a window \( W \) for which the parameters are \( c, u, \) and \( l \), where \( c \) is the size of the window, and \( u \) and \( l \) are the starting and ending points of the window, respectively. Initially, the values of \( c, u, \) and \( l \) are \( m/2, 1, \) and \( m/2 \), respectively. Window \( W \) can be defined according to Equation (8):

\[
(x_u, x_{u+1}, \ldots, x_l) \in W. \quad (8)
\]

Step 3: Slide the window \( W \) using a step size of one (i.e., increase the values of \( u \) and \( l \)), and calculate the accuracy of window \( P_w \) using Equation (9):

\[
P_w = \frac{\sum_{i=u}^{l} \mathbb{1}[p(x_i) = y_i]}{l - u}, \quad (9)
\]
where \( p(x_i) \) is the prediction \( x_i \), and \( y_i \) is the label of \( x_i \). If the accuracy is larger than the target accuracy \( TA \) and the value of \( c \) is not 2, then the window stops sliding.

Step 4: Reset the window parameters: the value of \( c \) is reduced by half, \( u \) does not change, and \( l = u + c - 1 \).

Step 5: Iterate Steps 3 and 4. Set the first instance in the window as threshold \( WT \) when \( c \) is equal to or lower than 2.

Step 6: Compare the confidence values of the instances with the threshold. If the confidence of the instance is larger than the threshold, then the instance produces a final prediction at the current level; otherwise, the instance is passed to the next level.

Unlike in the binning confidence screening approach, the threshold in window screening can be obtained by a self-adaptive method. Moreover, the performance of this approach is better than DBC-Forest, since the bin size is not adjusted. An illustrative example is given below.

**Example 3.** In this example, we set the target accuracy \( TA \) to 70\%, and employ the same 16 instances as in Example 2. The process of window screening is shown in Figure 6. In stage 1, we set the initial values to \( c = 8 \), \( u = 1 \), and \( l = 8 \). We know that the accuracy of the window is 0.875, which is greater than \( TA \). Thus, we slide the window \( W \) using a step size of one. We know that when \( u = 2 \) and \( l = 9 \), the accuracy of the window is also 0.875. However, when \( u = 3 \) and \( l = 10 \), the accuracy of the window is 0.625, which is less than \( TA \). We then enter stage 2, and reset the values to \( c = 4 \), \( u = 3 \), and \( l = 6 \). We know that the accuracy of this window is 0.75, which is greater than \( TA \). We then iterate this process until \( c = 4 \), \( u = 7 \), and \( l = 10 \). In this case, the accuracy of the window is 0.50, which is less than \( TA \). We then enter stage 3, and reset the values to \( c = 2 \), \( u = 7 \), and \( l = 8 \). In this case, the accuracy of the window is 1.0, which is greater than \( TA \). We iterate this process until \( c = 2 \), \( u = 8 \), and \( l = 9 \). In this case, the accuracy of the window is 0.50, which is less than \( TA \). Since \( c = 2 \), the window does not shrink further. Hence, the threshold is the confidence of instance 8, which is the same as for the case of \( k = 4 \) in Example 2.

### 4.3 HW-Forest

HW-Forest is an improved version of gcForest which consists of two parts: multi-grained scanning with hashing screening and cascade structure with window screening. Figure 7 illustrates the overall procedure of HW-Forest. In the first part, the instances are input into the multi-grained scanning stage. We then apply hashing screening to eliminate the redundant feature vectors, and the
Illustration of window screening. The size of window $c$ will be decreased when the accuracy of the window is lower than $T_{A}$. When the size of window is two and the accuracy is lower than $T_{A}$, the confidence of the first instance is selected as the threshold.

Framework of HW-Forest. A hashing screening strategy is adopted to eliminate redundant feature vectors, and window screening is applied to select high-confidence instances in the cascade forest. Remaining feature vectors are converted to class vectors using a random forest algorithm. The second part is a cascade forest structure with window screening mechanism to reduce the time costs and memory requirements. The class vectors in the form of the original data are input into the first level. A window screening mechanism is used to divide all instances into high- and low-confidence instances at each level. High-confidence instances are used to produce the final prediction at the current level, while low-confidence instances are passed to the next level.

Algorithm 1 shows pseudocode for HW-Forest.

5 EXPERIMENTS

In this section, we report the results of some experiments carried out to assess the performance of HW-Forest. We also demonstrate the effects of the hashing screening and window screening processes. This section is organized as follows. Section 5.1 introduces the experimental setting, including the parameters, experimental environment, and baseline methods. Section 5.2 investigates the effect of hashing screening with different window sizes. Section 5.3 validates that hashing screening can be adopted in other deep forest models. Section 5.4 discusses the influence of window screening. Section 5.5 compares the thresholds used in DBC-Forest and HW-Forest to validate the effect of window screening. Section 5.6 compares HW-Forest with other state-of-the-art models on benchmark datasets.

5.1 Experimental Setup

Parameter Settings. In all of our experiments, the models used the same parameters, which were as follows. For the cascade structure, each level was produced by five-fold cross-validation and had a random forest and a completely random forest. Each forest contained 50 decision trees. The
### Table 1. Summary of Datasets

| Name                  | Training | Testing | Features | Labels |
|-----------------------|----------|---------|----------|--------|
| MNIST                 | 56,000   | 14,000  | 784      | 10     |
| EMNIST                | 105,280  | 26,320  | 784      | 10     |
| FASHION-MNIST         | 56,000   | 14,000  | 784      | 10     |
| QMNIST                | 96,000   | 24,000  | 784      | 10     |
| CIFAR10               | 50,000   | 10,000  | 1,024    | 10     |
| ADULT                 | 39,074   | 9,768   | 14       | 2      |
| BANK (BANK MARKETING) | 32,950   | 8,238   | 20       | 2      |
| YEAST                 | 1,187    | 297     | 8        | 10     |
| LETTER                | 16,000   | 4,000   | 16       | 26     |
| IMDB                  | 40,000   | 10,000  | 5,000    | 2      |

---

**Algorithm 1: HW-Forest**

**Require:** Training set $S$, validation set $S_v$, and the maximal number of cascade levels $T$.

**Ensure:** HW-Forest model $f$.

1. Get $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_r)$ by scanning $S$;
2. Calculate $HT$ using the hashing screening mechanism;
3. Eliminate redundant feature vectors and obtain a new training set $S'$;
4. $t = 1, l_1 = 0, S_1 = S'$;
5. While $t \leq T$ do
6. $g = 99, c = \text{crad}(S)/2, u = 0, l = c$, and $l_0 = 1$;
7. Calculate $WT_t$ using the window screening mechanism;
8. $f_t = \text{prediction}(S')$;
9. Compute the validation error $l_t = f_t(S_v)$;
10. If $l_t > l_{t-1}$ then
11. Return $f$;
12. End if
13. $f = f_t$
14. $S_{t+1} = S_t \{x | x > WT_t, x \in S_t\}$
15. $t = t + 1$
16. End while
17. Return $f$;

---

number of cascade levels stopped increasing when the current level did not give an improvement in the accuracy of the previous level. For multi-grained scanning, we established three windows to scan the high-dimensional data, with sizes of $4 \times 4, 6 \times 6$, and $8 \times 8$. Each window had a random forest and a completely random forest, and the number of decision trees in these two forests was 30. $TA$ was set to decrease the error rate by 1/2.

**Datasets.** We used 10 benchmark datasets to verify the performance of HW-Forest. Table 1 summarizes the statistics of these datasets. Of these, MNIST, EMNIST, FASHION-MNIST, QMNIST, and CIFAR10 are commonly used high-dimensional datasets [34–36], while ADULT, BANK, YEAST, LETTER, and IMDB are low-dimensional datasets.

**Evaluation metrics and methodology.** In all of our experiments, we used the predictive accuracy as a measure of the classification performance, as it is suitable for balanced datasets, and the best accuracy were shown in bold in the corresponding tables. The training time was used to evaluate the efficiency, and the best training time was shown in bold. Statistical tests were also adopted as measurement criteria.

**Hardware.** In these experiments, we used a computer with $2 \times 2.20$ GHz CPUs (10 cores) and 128 GB main memory.
**Baseline methods.** To validate the performance of HW-Forest, we selected four state-of-the-art algorithms as competitive models: gcForest [9], gcForestcs [12], DBC-Forest [13], and AWDF [27]. To verify the effect of our hashing screening mechanism, we introduced this mechanism into gcForest, gcForestcs, DBC-Forest, and AWDF, to give H-gcForest, H-gcForestcs, HDBC-Forest, and H-AWDF, respectively.

**gcForest [9]:** This is a deep learning model that is inspired by deep neural networks, and adopts multi-grained scanning and a cascade structure to process the data.

**gcForestcs [12]:** This is an improved gcForest model in which a confidence screening mechanism is introduced to screen the instances at each level.

**DBC-Forest [13]:** This is an improved gcForestcs model. To increase the accuracy of gcForestcs, binning confidence screening is used to avoid the mispartitioning of instances at each level.

**AWDF [27]:** This is an adaptive weighted model in which adaptive weighting is used for every training instance at each level.

**H-gcForest, H-gcForestcs, HDBC-Forest, and H-AWDF:** To verify the influence of the hashing screening mechanism, we introduced it into gcForest, gcForestcs, DBC-Forest, and AWDF, to give H-gcForest, H-gcForestcs, HDBC-Forest, and H-AWDF, respectively.

**H-Forest (H-gcForest) and W-Forest:** To validate the influence of two screening mechanisms, we developed H-Forest (H-gcForest) and W-Forest; these are versions of HW-Forest without the window and hashing screening mechanisms, respectively.

To validate the performance of HW-Forest, we developed the following five research questions (RQ).

RQ1: What is the effect of hashing screening with different window sizes?
RQ2: What is the effect of hashing screening, and can it be adopted in other deep forest models?
RQ3: Compared with DBC-Forest, what is the effect of HW-Forest at each level?
RQ4: What is the difference between the thresholds in DBC-Forest and HW-Forest?
RQ5: Compared with state-of-the-art alternatives, what is the performance of HW-Forest?

To address RQ1, we selected 10 instances from FASHION-MINST to investigate the effect of hashing screening with different window sizes, and to explore the performance of HW-Forest with different window sizes. To answer RQ2, we introduced hashing screening into gcForest, gcForestcs, DBC-Forest, and AWDF, and compared their accuracies with the original models to verify the influence of hashing screening. In response to RQ3, we compared the accuracy and the number of instances between DBC-Forest and HW-Forest at each level. Specifically, we used MNIST, EMNIST, FASHION-MNIST, QMNIST, and IMDB in this experiment. To answer RQ4, we explored the thresholds used in the window screening and binning confidence screening mechanisms to show the difference. For RQ5, we compared the performance of HW-Forest with gcForest, gcForestcs, DBC-Forest, AWDF, H-Forest, and W-Forest.

### 5.2 Effect of Hashing Screening

In this section, we compared raw instances with instances processed by hashing screening, to explore the effect of the hashing screening mechanism. We conducted experiments with windows of size $2 \times 2$, $4 \times 4$, $6 \times 6$, and $8 \times 8$, and selected 10 instances from the FASHION-MNIST dataset. The results are shown in Figure 8.

Two observations can be made from these results.

1. The number of redundant feature vectors produced by a large window is significantly less than for a small window. For example, the number of redundant feature vectors in Figure 8(e) is 2, for a window size of $8 \times 8$; however, the number of redundant feature vectors in Figure 8(b) is 55, for a window size of $2 \times 2$. The reason for this is that on the FASHION-MNIST
dataset, the difference between the number of redundant feature vectors and important feature vectors is small. As the window size increases, the probability of containing important feature vectors increases, and thus the difference becomes smaller. Hence, with an increase in the window size, the number of redundant feature vectors decreases.

(2) Hashing screening reduces the number of redundant feature vectors but retains important information. For example, as shown in Figure 8(b), hashing screening retains the important features of instances, such as the toes of shoes and the sleeves of coats. The same phenomenon can be seen in the other figures. The reason for this is as follows. Since the important feature vectors are significantly different from each other, the distance between them is larger than for the redundant feature vectors. The redundant feature vectors are therefore eliminated and the important feature vectors are retained based on their distances.

To further explore the effect of the hashing screening mechanism, we conducted four experiments with windows of size $2 \times 2$, $4 \times 4$, $6 \times 6$, and $8 \times 8$. In each experiment, we compared the accuracy and time cost of W-gcForest and HW-gcForest for different window sizes. The results are shown in Table 2.

Based on these results, two observations can be made.

(1) With an increase in the window size, the time required for hash screening decreases. As shown in Table 2, this time is reduced by 227.08s (29.80%) and 56.83s (5.57%) when the window sizes are $2 \times 2$ and $6 \times 6$, respectively. The reason for this is as follows. As in the previous analysis, with an increase in the window size, the number of redundant feature vectors decreases, and hence the time required for hash screening decreases.
Table 2. Comparison of Performance between W-Forest and HW-gcForest for Different Window Sizes

| Window size | W-Forest | HW-Forest | Difference | W-Forest | HW-Forest | Difference |
|-------------|----------|-----------|------------|----------|-----------|------------|
| 2 × 2       | 89.98 ± 0.27 | 90.05 ± 0.18 | 0.07 | 989.19 ± 9.25 | 762.11 ± 10.32 | 227.08 (29.80%) |
| 4 × 4       | 90.57 ± 0.27 | 90.66 ± 0.10 | 0.09 | 1,049.75 ± 8.36 | 953.53 ± 9.68 | 96.22 (10.09%) |
| 6 × 6       | 90.28 ± 0.28 | 90.28 ± 0.28 | 0.00 | 1,077.05 ± 11.31 | 1,020.22 ± 11.81 | 56.83 (5.57%) |
| 8 × 8       | 98.98 ± 0.33 | 98.98 ± 0.33 | 0.00 | 1,192.44 ± 11.48 | 1,184.81 ± 13.19 | 7.63 (0.64%) |

Table 3. The Number of Eliminated Redundant Feature Vectors

| Dataset       | 2 × 2 | 4 × 4 | 6 × 6 | 8 × 8 |
|---------------|-------|-------|-------|-------|
|               | Total | Eliminated | Total | Eliminated | Total | Eliminated | Total | Eliminated |
| MNIST         | 196   | 105     | 169   | 72     | 144   | 49         | 121   | 23         |
| EMNIST        | 196   | 72      | 169   | 43     | 144   | 18         | 121   | 3          |
| FASHION-MNIST | 196   | 36      | 169   | 19     | 144   | 7          | 121   | 1          |
| QMNIST        | 196   | 105     | 169   | 72     | 144   | 49         | 121   | 23         |
| CIFAR10       | 256   | red1    | 225   | 1      | 196   | 1          | 169   | 0          |

(2) Hashing screening does not affect the accuracy of the model with different window sizes. For example, HW-Forest has the same accuracy as W-Forest for window sizes of 8 × 8 and 6 × 6, and has better accuracy than W-Forest when the window sizes are 4 × 4 and 2 × 2. This phenomenon indicates that using the distance of the feature vector as the criterion for eliminating redundant feature vectors can effectively identify redundant feature vectors with different window sizes. In other words, hashing screening is robust under different window sizes.

To further analyze the effect of hashing screening mechanism, we show the number of eliminated redundant feature vectors. Specifically, we adopted MNIST, EMNIST, FASHION-MNIST, QMNIST, and CIFAR10 datasets in the experiments, and conducted experiments with window of size 2 × 2, 4 × 4, 6 × 6, and 8 × 8. The results are shown in Table 3.

The experimental results give rise to the following observations.

(1) As the window size increases, the number of eliminated feature vectors decreases. For example, on the MNIST dataset, hashing screening eliminates 105 redundant feature vectors with window size 2 × 2. However, under the condition of window size 8 × 8, hashing screening eliminates only 23 redundant feature vectors. The same phenomenon can be seen in the other figures. The reason for this is as follows. Hashing screening eliminates redundant feature vectors by their distance, which is calculated by aHash algorithm. When the feature vectors are the same, aHash algorithm will generate the same binary sequences. Then, the feature vectors are eliminated, since their distance of binary sequences is 0. When the size of window increases, the number of features contained by feature vector increases, and the binary sequences with similar likelihood are reduced. Thus, the number of redundant feature vectors produced by big window size is lower than small window size.

(2) The number of eliminated feature vectors is different on different datasets. As shown in Table 3, on the QMNIST dataset, the number of eliminated feature vector is 72 with window size 2 × 2. However, on the CIFAR10 dataset, the number of eliminated feature vectors is only 1. The reason for this phenomenon is as follows. Hashing screening mechanism can adapt to different datasets. When the dataset has no or few redundant feature vectors, hashing screening can retain all important feature vectors which are non-redundant feature vectors.
To analyze the effect of the hashing screening mechanism, we conducted four experiments to compare the performance between pairs of models: gcForest and H-gcForest; gcForestcs and H-gcForestcs; and AWDF and H-AWDF. Experiments were conducted on the MNIST, EMNIST, FASHION-MNIST, QMNIST, and CIFAR10 datasets, since these need to be processed by multi-grained scanning. The comparison results are shown in Tables 4–8.

The experimental results give rise to the following observations.

(1) Hashing screening effectively decreases the time cost on the MNIST, EMNIST, FASHION-MNIST, and QMNIST datasets, but increases the time cost on the CIFAR10 dataset. For example, from Table 5, it can be seen that H-gcForestcs has a time cost that is 482.5 s (41.83%) lower than for gcForestcs on the MNIST dataset, but H-gcForestcs has a time cost that is 4353.0 s (2.83%) higher than for gcForestcs on the CIFAR10 dataset. This phenomenon can also be seen in the other tables. The reason for this is as follows. According to Table 3 in the previous section, we know that hashing screening can find more redundant features on

### Table 4. Comparison of Performance between gcForest and H-gcForest

| Dataset   | Accuracy (%) | Time(s) |
|-----------|--------------|---------|
|           | gcForest     | H-gcForest | Difference | gcForest     | H-gcForest | Difference |
| MNIST     | 98.77 ± 0.17 | 98.75 ± 0.12 | -0.02 | 1972.27 ± 6.39 | 1464.86 ± 11.54 |
| EMNIST    | 86.18 ± 0.24 | 86.13 ± 0.28 | -0.05 | 8774.77 ± 108.08 | 6920.12 ± 51.39 |
| FASHION-MNIST | 89.94 ± 0.29 | 90.05 ± 0.29 | 0.09 | 2411.10 ± 10.28 | 2272.63 ± 25.18 |
| QMNIST    | 98.93 ± 0.06 | 98.92 ± 0.06 | -0.01 | 2332.79 ± 31.80 | 1790.10 ± 17.71 |
| CIFAR10   | 53.49 ± 0.43 | 53.56 ± 0.53 | 0.07 | 4355.52 ± 171.66 | 4366.91 ± 57.35 |

### Table 5. Comparison of Performance between gcForestcs and H-gcForestcs

| Dataset   | Accuracy (%) | Time(s) |
|-----------|--------------|---------|
|           | gcForestcs   | H-gcForestcs | Difference | gcForestcs   | H-gcForestcs | Difference |
| MNIST     | 98.20 ± 0.09 | 98.19 ± 0.12 | -0.01 | 1636.00 ± 13.47 | 1153.30 ± 8.65 |
| EMNIST    | 87.11 ± 0.23 | 87.13 ± 0.30 | 0.02 | 5945.37 ± 159.68 | 513.32 ± 128.73 |
| FASHION-MNIST | 89.94 ± 0.29 | 90.03 ± 0.30 | 0.07 | 1943.09 ± 6.50 | 1850.41 ± 8.41 |
| QMNIST    | 98.40 ± 0.12 | 98.41 ± 0.12 | 0.01 | 1991.11 ± 17.71 | 1454.08 ± 14.77 |
| CIFAR10   | 54.62 ± 0.23 | 54.52 ± 0.19 | -0.10 | 4233.23 ± 65.18 | 4353.06 ± 29.94 |

### Table 6. Comparison of Performance between DBC-Forest and HDBC-gcForest

| Dataset   | Accuracy (%) | Time(s) |
|-----------|--------------|---------|
|           | DBC-Forest   | HDBC-gcForest | Difference | DBC-Forest   | HDBC-gcForest | Difference |
| MNIST     | 99.03 ± 0.07 | 99.01 ± 0.06 | -0.02 | 1636.10 ± 17.79 | 1159.99 ± 9.90 |
| EMNIST    | 86.82 ± 0.26 | 86.83 ± 0.16 | -0.01 | 7226.23 ± 112.34 | 6349.11 ± 162.52 |
| FASHION-MNIST | 90.57 ± 0.29 | 90.70 ± 0.20 | 0.13 | 2056.95 ± 13.38 | 1928.22 ± 13.91 |
| QMNIST    | 99.08 ± 0.12 | 99.05 ± 0.06 | -0.03 | 1979.00 ± 22.14 | 1467.99 ± 21.23 |
| CIFAR10   | 54.79 ± 0.56 | 54.82 ± 0.34 | 0.03 | 4324.68 ± 121.53 | 4366.26 ± 66.92 |

### Table 7. Comparison of Performance between AWDF and H-AWDF

| Dataset   | Accuracy (%) | Time(s) |
|-----------|--------------|---------|
|           | AWDF         | H-AWDF | Difference | AWDF         | H-AWDF | Difference |
| MNIST     | 98.80 ± 0.11 | 98.80 ± 0.11 | 0.00 | 1697.79 ± 18.36 | 1192.30 ± 11.95 |
| EMNIST    | 86.74 ± 0.18 | 86.70 ± 0.25 | -0.04 | 7385.44 ± 103.03 | 6664.62 ± 201.76 |
| FASHION-MNIST | 89.89 ± 0.27 | 90.05 ± 0.14 | 0.06 | 1821.65 ± 15.30 | 1734.06 ± 16.83 |
| QMNIST    | 98.93 ± 0.08 | 98.95 ± 0.10 | 0.02 | 1988.55 ± 66.69 | 1458.53 ± 14.06 |
| CIFAR10   | 54.08 ± 0.38 | 54.12 ± 0.51 | 0.04 | 4537.44 ± 220.95 | 4685.58 ± 193.6 |

### 5.3 Influence of Hashing Screening

To analyze the effect of the hashing screening mechanism, we conducted four experiments to compare the performance between pairs of models: gcForest and H-gcForest; gcForestcs and H-gcForestcs; and AWDF and H-AWDF. Experiments were conducted on the MNIST, EMNIST, FASHION-MNIST, QMNIST, and CIFAR10 datasets, since these need to be processed by multi-grained scanning. The comparison results are shown in Tables 4–8.

The experimental results give rise to the following observations.

(1) Hashing screening effectively decreases the time cost on the MNIST, EMNIST, FASHION-MNIST, and QMNIST datasets, but increases the time cost on the CIFAR10 dataset. For example, from Table 5, it can be seen that H-gcForestcs has a time cost that is 482.5 s (41.83%) lower than for gcForestcs on the MNIST dataset, but H-gcForestcs has a time cost that is 4353.0 s (2.83%) higher than for gcForestcs on the CIFAR10 dataset. This phenomenon can also be seen in the other tables. The reason for this is as follows. According to Table 3 in the previous section, we know that hashing screening can find more redundant features on
Table 8. Comparison of Performance between W-Forest and HW-Forest

| Dataset       | Accuracy (%) | Time(s)       |
|---------------|--------------|---------------|
|               | W-Forest     | HW-Forest     | Difference | W-Forest     | HW-Forest     | Difference |
| MNIST         | 99.05 ± 0.08 | 99.07 ± 0.09  | −0.02      | 1648.92 ± 20.56 | 1135.42 ± 9.32 | 513.5 (45.23%) |
| EMNIST        | 87.24 ± 0.18 | 87.22 ± 0.17  | −0.02      | 6856.64 ± 80.16 | 6408.14 ± 65.33 | 448.5 (7.00%)  |
| FASHION-MNIST | 90.77 ± 0.34 | 90.79 ± 0.24  | 0.02       | 1990.47 ± 13.91 | 1404.17 ± 22.82 | 586.30 (4.53%) |
| QMNIST        | 99.13 ± 0.05 | 99.13 ± 0.05  | 0.00       | 1965.98 ± 34.68 | 1743.09 ± 12.00 | 542.89 (33.46%)|
| CIFAR10       | 55.18 ± 0.41 | 55.13 ± 0.05  | −0.05      | 4303.07 ± 29.08 | 4358.68 ± 35.78 | −55.51 (−1.29%)|

Table 9. Significance Testing for Time Cost

| DATASET     | H-gForest | H-gForestcs | HDBC-Forest | H-AWDF | HW-Forest |
|-------------|-----------|-------------|-------------|--------|-----------|
| MNIST       | R (192.15)| R (32.08)   | R (42.58)   | R (44.3)| R (44.30) |
| EMNIST      | R (28.02) | R (6.56)    | R (6.83)    | R (5.99)| R (5.36)  |
| FASHION-MNIST | R (12.58)| R (35.44)   | R (11.49)   | R (7.71)| R (7.71)  |
| QMNIST      | R (25.45) | R (44.35)   | R (34.31)   | R (28.76)| R (28.76)|
| CIFAR10     | A (0.12)  | A (1.1)     | A (0.69)    | A (0.97)| A (1.01)  |

the MNIST, EMNIST, FASHION-MNIST, and QMNIST datasets, but less redundant features on the CIFAR10 dataset. As the analysis of time complexities of hashing screening and HW-Forest in Section 4.1, we know that the greater the number of redundant features, the more training time will be reduced. The experimental results are consistent with the theoretical analysis, which also verifies the correctness of the theoretical analysis.

(2) Hashing screening does not influence the accuracy of the models. The results of 25 experiments are presented in Tables 4–8, and it can be seen that the models with hashing screening have better accuracy than the original models in 12 experiments, worse accuracy than the original models in 11 experiments, and the same as the original models in 2 experiments. For example, Table 5 shows that H-gcForestcs has better accuracy than gcForestcs on three datasets (EMNIST, FASHION-MNIST, and QMNIST) and worse accuracy than gcForestcs on MNIST. The reason for this is as follows. Since the important feature vectors have larger distances than the redundant feature vectors, hashing screening effectively screens out redundant feature vectors based on their distances. More importantly, these redundant feature vectors make lower contributions to the classification accuracy. Hence, the models using hashing screening have almost the same accuracy as the original models.

To further evaluate the influence of hashing screening, we applied a t-test for statistical hypothesis test.

To measure the influence of the hashing screening mechanism, we applied a t-test to the time cost and accuracy. More specifically, we compared the accuracy and time cost of the models at each fold, and obtained different values \((D_1, D_2, \ldots, D_k)\). We then calculated the statistics according to \(|\sqrt{k} \times \mu/\sigma|\), where \(k\) is the number of cross-validation folds, \(\mu\) is the mean of \((D_1, D_2, \ldots, D_k)\), and \(\sigma\) is the standard deviation of \((D_1, D_2, \ldots, D_k)\). The results for the time cost and accuracy are shown in Tables 9 and 10, respectively.

We can reject (R) or accept (A) the null hypothesis based on the statistics. For example, for the MNIST dataset in Table 10, the accuracy of H-gcForest is not significantly different from that of gcForest, since \(T_{0.05,4}\) is 2.13, which is larger than the \(t\)-test statistic of 0.88. Furthermore, for the MNIST dataset, Table 9 shows that the time costs of H-Forest and HW-Forest are significantly different, since the \(t\)-test statistic is 192.15, which is much larger than 2.13. The results of these statistical hypothesis tests show that hashing screening effectively reduces the time cost by eliminating redundant feature vectors, without influencing the accuracy. More importantly, when used
with other deep forest models, hashing screening can achieve the same accuracy as the original models.

Based on the above experimental results and statistical analysis, we conclude that hashing screening can reduce the time cost without influencing the accuracy.

5.4 Influence of Window Screening

To validate the effect of window screening, we observed the accuracy and the number of instances at each level. We selected DBC-Forest as a competitive algorithm and used the MNIST, EMNIST, FASHION-MNIST, QMNIST, and IMDB datasets, since our model and competitive algorithm produced more layers for these datasets. Comparisons of the accuracy and numbers of instances for MNIST, EMNIST, FASHION-MNIST, QMNIST, and IMDB are shown in Figures 9–13, respectively. These results give rise to the following observations.

(1) HW-Forest achieves higher accuracy than DBC-Forest at each level. For example, it can be seen from Figure 13 that for the IMDB dataset, the accuracy of HW-Forest is 0.09% higher than that of DBC-Forest.

Table 10. Significance Testing for Accuracy

| DATASET     | H-gcForest | H-gcForestcs | HDBC-Forest | H-AWDF | HW-Forest |
|-------------|------------|--------------|-------------|--------|-----------|
| MNIST       | A (0.88)   | A (1.52)     | A (0.47)    | A (1.18)| A (0.60)  |
| EMNIST      | A (0.08)   | A (1.99)     | A (0.13)    | A (0.98)| A (0.75)  |
| FASHION-MNIST| A (1.15)  | A (1.89)     | A (0.80)    | A (1.17)| A (0.09)  |
| QMNIST      | A (0.87)   | A (0.33)     | A (1.13)    | A (0.40)| A (0.84)  |
| CIFAR10     | A (0.78)   | A (1.35)     | A (1.35)    | A (0.24)| A (1.2)   |

Fig. 9. Comparison of accuracy and number of instances for the MNIST dataset.

Fig. 10. Comparison of accuracy and number of instances for the EMNIST dataset.
than DBC-Forest at the second level, and 0.19% higher than DBC-Forest at the last level. The reason for this is that window screening is a self-adaptive mechanism, and produces more reasonable thresholds than the binning screening mechanism without the need for hyperparameter tuning. Hence, the instances screened by HW-Forest have higher accuracy than those screened by DBC-Forest at each level.

(2) HW-Forest has a lower number of instances than DBC-Forest at each level. For example, on the MNIST dataset, HW-Forest has 1,213 fewer instances than DBC-Forest at the second level, and 146 fewer instances than DBC-Forest at the last level. The reason for this is that since the performance of DBC-Forest depends on the hyperparameters, it is not easy to find high-confidence instances. However, HW-Forest adopts a self-adaptive mechanism to precisely select high-confidence instances.
5.5 Effect of Window Screening

In this section, we explore the difference between window screening and binning confidence screening. To demonstrate the performance of our approach on different types of datasets, we used MNIST, IMDB, LETTER, and BANK. Comparisons of the thresholds for the MNIST, IMDB, LETTER, and BANK datasets are shown in Figures 14–17, respectively.

As shown in Figures 14–17, HW-Forest gives a more reasonable threshold than DBC-Forest. For example, on the MNIST dataset, DBC-Forest cannot avoid accuracy fluctuations, and some high-confidence instances screened with this threshold have low accuracy. Using HW-Forest, the screened high-confidence instances have significantly higher accuracy. The reason for this is that unlike the dependence of DBC-Forest on hyperparameter tuning, HW-Forest selects instances at each level without the need for hyperparameter tuning. Hence, HW-Forest gives a more reasonable threshold than DBC-Forest.
5.6 Performance of HW-Forest

In this section, to validate the accuracy of HW-Forest, we used gcForest, gcForestcs, DBC-Forest, AWDF, H-Forest, and W-Forest as competitive algorithms. Experiments were conducted on 10 benchmark datasets, and comparisons of the results for accuracy and time cost are shown in Tables 11 and 12, respectively.

According to the experimental results, we make the following observations.

(1) As shown in Table 11, HW-Forest achieves the highest accuracy on most datasets. For example, on the QMNIST dataset, HW-Forest achieves the highest accuracy of 99.07%. The same phenomenon can be found for the other datasets. The reason for this is that HW-Forest uses window screening to improve the accuracy of DBC-Forest, and this produces more reasonable thresholds than the binning screening mechanism. Hence, HW-Forest achieves higher accuracy than DBC-Forest, and also achieves higher accuracy than other models. As shown in Table 12, for high-dimensional datasets, HW-Forest runs
Table 11. Comparison of Accuracy (%)

| DATASET | gcforest | gcForestcs | DBC-Forest | AWDF | H-Forest | W-Forest | HW-Forest |
|---------|----------|------------|------------|------|----------|----------|-----------|
| MNIST   | 98.77 ± 0.12 | 98.20 ± 0.09 | 99.03 ± 0.07 | 98.80 ± 0.11 | 98.75 ± 0.12 | 99.05 ± 0.08 | 99.07 ± 0.09 |
| EMNIST  | 86.18 ± 0.24 | 87.11 ± 0.23 | 86.82 ± 0.26 | 86.74 ± 0.18 | 86.13 ± 0.28 | 87.24 ± 0.18 | 87.22 ± 0.17 |
| FASHION-MNIST | 89.94 ± 0.29 | 89.94 ± 0.29 | 90.57 ± 0.22 | 89.89 ± 0.27 | 90.05 ± 0.29 | 90.77 ± 0.29 | 90.79 ± 0.24 |
| QMNIST  | 98.93 ± 0.06 | 98.40 ± 0.12 | 99.08 ± 0.08 | 98.93 ± 0.08 | 98.92 ± 0.06 | 99.13 ± 0.05 | 99.15 ± 0.03 |
| CIFAR10 | 53.49 ± 0.43 | 54.62 ± 0.23 | 54.79 ± 0.56 | 54.08 ± 0.3 | 53.56 ± 0.53 | 55.18 ± 0.41 | 55.13 ± 0.39 |
| ADULT   | 85.99 ± 0.37 | 86.04 ± 0.35 | 86.11 ± 0.29 | 85.86 ± 0.15 | 85.99 ± 0.37 | 86.18 ± 0.65 | 86.18 ± 0.65 |
| BANK    | 91.43 ± 0.23 | 91.58 ± 0.19 | 91.62 ± 0.16 | 91.45 ± 0.23 | 91.43 ± 0.23 | 91.60 ± 0.22 | 91.60 ± 0.22 |
| YEAST   | 62.06 ± 3.77 | 62.00 ± 3.50 | 62.13 ± 3.70 | 62.53 ± 4.10 | 62.06 ± 3.77 | 62.67 ± 2.07 | 62.67 ± 2.07 |
| IMDB    | 88.81 ± 0.10 | 88.88 ± 0.14 | 89.39 ± 0.32 | 88.94 ± 0.23 | 88.81 ± 0.10 | 89.58 ± 0.26 | 89.58 ± 0.26 |
| LETTER  | 97.02 ± 0.22 | 96.83 ± 0.27 | 97.07 ± 0.23 | 96.65 ± 0.18 | 97.02 ± 0.22 | 97.19 ± 0.16 | 97.19 ± 0.16 |

Table 12. Comparison of Time Cost (s)

| DATASET | gcforest | gcForestcs | DBC-Forest | AWDF | H-Forest | W-Forest | HW-Forest |
|---------|----------|------------|------------|------|----------|----------|-----------|
| MNIST   | 1972.27 ± 8.39 | 1636.00 ± 13.47 | 1636.10 ± 17.79 | 1697.79 ± 18.36 | 1644.86 ± 11.54 | 1648.92 ± 20.56 | 1135.42 ± 9.32 |
| EMNIST  | 8774.77 ± 108.08 | 5945.37 ± 159.68 | 7926.23 ± 112.34 | 7385.44 ± 103.03 | 6920.12 ± 51.39 | 6856.64 ± 80.16 | 6408.14 ± 65.33 |
| FASHION-MNIST | 2411.10 ± 10.28 | 1943.09 ± 6.50 | 2056.95 ± 13.38 | 1821.65 ± 15.30 | 2272.63 ± 25.18 | 1990.47 ± 13.91 | 1904.17 ± 22.82 |
| QMNIST  | 2332.79 ± 31.80 | 1991.11 ± 17.71 | 1979.00 ± 22.14 | 1998.55 ± 66.69 | 1790.10 ± 17.71 | 1965.98 ± 34.68 | 1473.09 ± 12.00 |
| CIFAR10 | 4555.52 ± 171.66 | 4233.23 ± 65.18 | 4324.68 ± 121.53 | 4537.44 ± 220.95 | 4366.91 ± 57.35 | 4303.07 ± 29.08 | 4538.679 ± 35.78 |
| ADULT   | 10.63 ± 0.18 | 9.88 ± 0.08 | 10.18 ± 0.23 | 10.21 ± 0.12 | 10.63 ± 0.18 | 10.32 ± 0.18 | 10.15 ± 0.08 |
| BANK    | 12.68 ± 0.13 | 9.83 ± 0.26 | 10.84 ± 0.07 | 11.64 ± 0.06 | 12.77 ± 0.16 | 11.86 ± 0.11 | 12.05 ± 0.12 |
| YEAST   | 8.52 ± 0.04 | 8.47 ± 0.07 | 8.56 ± 0.09 | 10.65 ± 0.23 | 8.52 ± 0.04 | 8.69 ± 0.03 | 8.65 ± 0.33 |
| IMDB    | 1257.24 ± 6.28 | 475.65 ± 10.53 | 753.52 ± 12.18 | 1109.72 ± 16.22 | 1257.24 ± 6.28 | 551.96 ± 8.41 | 551.96 ± 8.41 |
| LETTER  | 11.89 ± 0.18 | 9.02 ± 0.10 | 9.02 ± 0.10 | 10.68 ± 0.12 | 11.89 ± 0.18 | 9.63 ± 0.14 | 9.63 ± 0.14 |

faster than the competitive models in most cases; for low-dimensional datasets, HW-Forest runs slower than gcForestcs, but faster than the other competitive models. For example, on the MNIST dataset, the time cost for HW-gcForest is 1135.42 s, which is faster than all other models. On the ADULT dataset, the time cost of HW-gcForest is 10.15 s, which is slower than gcForestcs but faster than the other competitive models. The reason for this is that for high-dimensional datasets, deep forest models take a long time to find the classification models. If we can reduce the number of feature vectors, the running time can be significantly reduced. HW-Forest employs hashing screening to effectively eliminate redundant feature vectors, and therefore runs faster than the competitive models for high-dimensional datasets. However, for low-dimensional datasets, the deep forest models take only a short time to find the classification models, and these datasets do not need to be processed by multi-grained scanning. Hence, the running time of precisely selecting instances at each level cannot be neglected, and HW-Forest runs slower than gcForestcs.

(2) HW-Forest and W-Forest achieve the same accuracy on most datasets. For example, from Table 11, we can see that for the QMNIST dataset, HW-Forest and W-Forest achieve the same accuracy of 99.13%. The same effect can be seen for the ADULT, BANK, YEAST, IMDB, and LETTER datasets. For the MNIST, FASHION-MNIST, and EMNIST datasets, the results for the accuracy are close. The reason for this is that low-dimensional datasets do not need to be processed by multi-grained scanning. The results of HW-Forest are therefore the same as those of W-Forest, since W-Forest is simply the HW-Forest model without the hashing screening mechanism. For high-dimensional datasets, as demonstrated in our analysis of the effect of hashing screening in Section 5.3, the hashing screening mechanism does not influence the accuracy of the model. Hence, the accuracies of HW-Forest and W-Forest are almost the same.

To further evaluate the performance of HW-Forest, we applied Friedman and Nemenyi tests [37, 38] for statistical hypothesis test.

(1) Friedman test: This test ranks the models according to their accuracy, where the highest is given a score of one, the second highest has a score of two, and so on. The mean values...
Fig. 18. Results of a Nemenyi test, where the dots show the mean value for each model, and the horizontal bar across each dot denotes the critical difference.

For the gcForest, gcForestcs, DBC-Forest, AWDF, HW-Forest, H-Forest, and W-Forest models were 6, 5.2, 3, 5.1, 5.5, 1.6, and 1.61, respectively. We then calculated the Friedman test statistic according to \((N-1) \times \tau_{x^2}/(N \times (k-1) - \tau_{x^2})\), where \(\tau_{x^2} = 12 \times N \times (\sum_{i=1}^{k} r_i^2 - k \times (k+1)^2/4)/k \times (k+1)\), and \(N\) and \(k\) are the number of datasets and models, respectively. The Friedman statistic was 29.3, which was larger than \(T_{0.05,4} = 2.132\). Hence, we reject the null hypothesis.

(2) Nemenyi test: To further compare the models, we adopted the Nemenyi test to calculate the critical difference value. The result was 2.601, according to the formula \(q_{0.1,7} \times k \times (k+1)/6 \times N\), where \(q_{0.1,7} = 2.693\). As shown in Figure 18, the accuracy of HW-Forest is the same as for W-Forest, which indicates that HW-Forest is statistically consistent with W-Forest. More importantly, the accuracy of HW-Forest is significantly higher than the other models except W-Forest, meaning that HW-Forest statistically outperforms the other competitive models.

6 CONCLUSION AND FUTURE WORK

To improve the performance and efficiency of the deep forest model, we propose a novel deep forest model called HW-Forest in which two mechanisms are applied to improve the performance: hashing screening and window screening. Hashing screening employs the feature distance as the criterion for selecting feature vectors, and this is calculated using a perceptual hashing algorithm. The number of redundant feature vectors can therefore be reduced by hashing screening. We also propose a window screening mechanism which produces the threshold by using a self-adaptive process for the length of the window. This method can effectively find the threshold without using an unsuitable hyperparameter for the size of the bins. Our experimental results demonstrate that hashing screening effectively reduces the time cost of multi-grained scanning, and the window screening mechanism outperforms the binning confidence screening mechanism. More importantly, our experimental results indicate that HW-Forest gives better performance than other competitive models, since it is statistically significantly better than these models.

In this article, we investigate an improved deep forest model called HW-Forest, which employs hashing screening and window screening mechanisms. Experimental results validate that HW-Forest outperforms other state-of-the-art deep forest models. However, we notice that all deep forest models have the following weaknesses.

(1) Deep forest models have better performance than other state-of-the-art deep learning models on MNIST, EMNIST, FASHION-MNIST, and QMNIST, but worse performance than other
state-of-the-art deep learning models on CIFAR10. For example, CNN achieves best test accuracy 98.17% on MNIST, 79.61% on EMNIST, 84.10% on FASHION-MNIST, and 97.82% on QMNIST. These results are inferior to HW-Forest: 99.07% on MNIST, 87.22% on EMNIST, 90.79% on FASHION-MNIST, and 99.13% on QMNIST [39]. However, HW-Forest achieves test accuracy 55.13% on CIFAR10 which is significantly worse than ResNet-18 91.98% [40]. We know that MNIST, EMNIST, FASHION-MNIST, and QMNIST are 28 × 28 grayscale images datasets, while CIFAR10 is a 32 × 32 sized color images dataset. The above results indicate that Deep forest models can achieve better performance on gray images. How to obtain better results on color images is worth exploring.

(2) Another disadvantage of deep forest models is that its space complexity is large, which means that it consumes a lot of memory in the training process. Thus, it is difficult for deep forest models to obtain the training model on large-scale datasets. All improved models based on deep forest (including HW-Forest) have the same problem. How to reduce the space complexity to obtain the training model on large-scale datasets is worthy of further research.

REFERENCES

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

[2] Shuhui Cheng, Youxi Wu, Yan Li, Fang Yao, and Fan Min. 2021. TWD-SFNN: Three-way decision with single hidden layer feedforward neural network. *Information Sciences* 579 (2021), 15–32.

[3] Youxi Wu, Lanfang Luo, Yan Li, Lei Guo, Philippe Fournier-Viger, Xingquan Zhu, and Xindong Wu. 2022. NTP-Miner: Nonoverlapping three-way sequential pattern mining. *ACM Transactions on Knowledge Discovery from Data* 16, 3 (2022), 51.

[4] Ziyang Zhang, Chenrui Duan, Tao Lin, Shoujun Zhou, Yuanquan Wang, and Xuedong Gao. 2020. GVFOM: A novel external force for active contour based image segmentation. *Information Sciences* 506, C (2020), 1–18.

[5] Zixing Zhang, Jürgen Geiger, Jouni Pohjalainen, Amr El-Desoky Mousa, Wenyu Jin, and Björn Schuller. 2018. Deep learning for environmentally robust speech recognition: An overview of recent developments. *ACM Transactions on Intelligent Systems and Technology* 9, 5 (2018), 49.

[6] Jipeng Qiang, Zhenyu Qian, Yun Li, Yunhao Yuan, Xindong Wu. 2022. Short text topic modeling techniques, applications, and performance: A survey. *IEEE Transactions on Knowledge and Data Engineering* 34, 3 (2022), 1427–1445.

[7] Dong Liu, YouXi Wu, and He Jiang. 2016. FP-ELM: An online sequential learning algorithm for dealing with concept drift. *Neurocomputing* 207, C, 322–334.

[8] YouXi Wu, Dong Liu, and He Jiang. 2017. Length-changeable incremental extreme learning machine. *Journal of Computer Science and Technology* 32, 3 (2017), 630–643.

[9] Zhihua Zhou and Ji Feng. 2019. Deep forest. *National Science Review* 6, 1 (2019), 74–86.

[10] Ran Su, Xinyi Liu, Leyi Wei, and Quan Zou. 2019. Deep-Resp-Forest: A deep forest model to predict anti-cancer drug response. *Methods* 166 (2019), 91–102.

[11] Feng Yang, Qizhi Xu, Bo Li, and Yan Ji. 2018. Ship detection from thermal remote sensing imagery through region-based deep forest. *IEEE Geoscience & Remote Sensing Letters* 15, 3 (2018), 449–453.

[12] Ming Pang, Kaiming Ting, Peng Zhao, and Zhihua Zhou. 2020. Improving deep forest by screening. *IEEE Transactions on Knowledge and Data Engineering* (2020). DOI: 10.1109/TKDE.2020.3038799

[13] Pengfei Ma, Youxi Wu, Yan Li, Lei Guo, and Zhao Li. 2022. DBC-Forest: Deep forest with binning confidence screening. *Neurocomputing* 475, C (2022), 112–122.

[14] Liang Sun, Zhanhao Mo, Fuhua Yan, Liming Xia, Fei Shan, Zhongxiang Ding, Bin Song, Wanchun Gao, Wei Shao, Feng Shi, Huan Yuan, Huiting Jiang, Dijia Wu, Ying Wei, Yaozong Gao, He Sui, Daqiang Zhang, and Dinggang Shen. 2020. Adaptive feature selection guided deep forest for covid-19 classification with chest CT. *IEEE Journal of Biomedical and Health Informatics* 24, 10 (2020), 2798–2805.

[15] Yang Guo, Shuhui Liu, Zhanhui Li, and Xuequn Shang. 2018. BCDForest: A boosting cascade deep forest model towards the classification of cancer subtypes based on gene expression data. *BMC Bioinformatics* 19, S5 (2018), 1–13.

[16] Yunyun Dong, Wenkai Yang, Jiawen Wang, Juanjuan Zhao, and Yan Qiang. 2019. MLW-gcForest: A multi-weighted gcForest model for cancer subtype classification by methylation data. *Applied Sciences* 9, 17 (2019), 3589.

[17] Shouxiang Wang, Haiwen Chen, Luyang Guo, and Di Xu. 2021. Non-intrusive load identification based on the improved voltage-current trajectory with discrete color encoding background and deep-forest classifier. *Energy and Buildings* 244, 2 (2021), 111043.
[18] Juan Cheng, Meiyao Chen, Chang Li, Yu Liu, Rencheng Song, Aiping Liu, and Xun Chen. 2020. Emotion recognition from multi-channel EEG via deep forest. IEEE Journal of Biomedical and Health Informatics 25, 2 (2020), 453–464.

[19] Yinfei Fang, Haiyang Yang, Xuguang Zhang, Han Liu, and Bo Tao. 2021. Multi-feature input deep forest for EEG-based emotion recognition. Frontiers in Neurorobotics 14 (2021), 617531.

[20] Liu Zhang, Heng Sun, Zhenhong Rao, and Haiyan Ji. 2020. Hyperspectral imaging technology combined with deep forest model to identify frost-damaged rice seeds. Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy 229 (2020), 117973.

[21] Kheir Eddine Daouadi, Rim Zghal Rebaï, and Ikram Amous. 2021. Optimizing semantic deep forest for tweet topic classification. Information Systems 101, 2 (2021), 101801.

[22] Meng Zhou, Xinhua Zeng, and Aozhu Chen. 2019. Deep forest hashing for image retrieval. Pattern Recognition 95, C (2019), 114–127.

[23] Chao Ma, Zhenbing Liu, Zhiguang Cao, Wen Song, Jie Zhang, and Weiliang Zeng. 2020. Cost-sensitive deep forest for price prediction. Pattern Recognition 107 (2020), 107499.

[24] Shenhuan Lyu, Liang Yang, and Zhihua Zhou. 2019 A refined margin distribution analysis for forest representation learning. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vol. 32. 5530–5540.

[25] Ludovic Arnould, Claire Boyer, and Erwan Scornet. 2021. Analyzing the tree-layer structure of Deep Forests. In Proceedings of the 38th International Conference on Machine Learning, Vol. 139. 342–350.

[26] Yihe Cheng, Shenhuan Lyu, and Yuan Jiang. 2021. Improving deep forest by exploiting high-order interactions. In Proceedings of the 21th IEEE International Conference on Data Mining. 1036–1041.

[27] Lev V. Utkin, Andrei V. Konstantinov, Viacheslav S. Chukanov, and Anna A. Meldo. 2020. A new adaptive weighted deep forest and its modifications. International Journal of Information Technology & Decision Making 19, 4 (2020), 963–986.

[28] Liang Yang, Xizhu Wu, Yuan Jiang, and Zhihua Zhou. 2020. Multi-label learning with deep forest. In Proceedings of the European Conference on Artificial Intelligence, Vol. 325. 1634–1641.

[29] Zheng Zhang, Xiaofeng Zhu, Guangming Lu, and Yudong Zhang. 2021. Probability ordinal-preserving semantic hashing for large-scale image retrieval. ACM Transactions on Knowledge Discovery from Data 15, 3 (2021), 1–22. DOI: 10.1145/3442204

[30] Huifang Yang, Chenghao Tu, and Chusong Chen. 2021. Learning binary hash codes based on adaptable label representations. IEEE Transactions on Neural Networks and Learning Systems (2021). DOI: 10.1109/TVNNSL.2021.3095399

[31] Yi Xu, Xianglong Liu, Binshuai Wang, Renshuai Tao, Ke Xia, and Xianbin Cao. 2021. Fast nearest subspace search via random angular hashing. IEEE Transactions on Multimedia 23 (2021), 342–352.

[32] Wen Fang, Haimiao Hu, Zihao Hu, Shengcai Liao, and Bo Li. 2018. Perceptual hash-based feature description for person re-identification. Neurocomputing 272 (2018), 520–531.

[33] Breiman Leo. 2001. Random forests. Machine Learning 45, 1 (2001), 5–32.

[34] Kafeng Wang, Haoyi Xiong, Jiang Bian, Zhanxing Zhu, Qian Gao, and Zhishan Guo. 2021. Sampling sparse representations with randomized measurement langevin dynamics. ACM Transactions on Knowledge Discovery from Data 15, 2 (2021), 1–21. DOI: 10.1145/3427585

[35] Yong Liu, Shizhong Liao, Shali Jiang, Lizhong Ding, Hailun Lin, and Weiying Wang. 2020. Fast cross-validation for kernel-based algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence 42, 5 (2020), 1083–1096.

[36] Shaoxing Zeng, Bob Zhang, Jianping Gou, Yuehua Wang, and Wei Huang. 2021. Fast and robust dictionary-based classifier for image data. ACM Transactions on Knowledge Discovery from Data 15, 6 (2021), 1–22. DOI: 10.1145/3449360

[37] Youxi Wu, Yuehua Wang, Yan Li, Xingquan Zhu, and Xindong Wu. 2021. Top-k self-adaptive contrast sequential pattern mining. IEEE Transactions on Cybernetics. DOI: 10.1109/TCYB.2021.3082114

[38] Youxi Wu, Meng Geng, Yan Li, lei Guo, Zhao Li, Philippe Fournier-Viger, Xingquan Zhu, and Xindong Wu. 2021. HANP-Miner: High average utility nonoverlapping sequential pattern mining. Knowledge-Based Systems, 229, C (2021), 107361.

[39] H. M. Dipu Kabir, Moloud Abdar, Seyed Mohammad Jafar Jalali, Abbas Khosravi, Amir F. Atiya, Saeid Nahavandi, and Dipti Srinivasan. 2020. SpinalNet: Deep neural network with gradual input. arXiv:2007.03347. Retrieved from https://arxiv.org/abs/2007.03347.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 770–778.