Multi-Feature Broad Learning System for Image Classification

Article *in* International Journal of Pattern Recognition and Artificial Intelligence · September 2021

DOI: 10.1142/S0218001421500336

7 authors, including:

Ran Liu
Chongqing University
43 PUBLICATIONS **109** CITATIONS
SEE PROFILE

Yaqiong Liu
Chongqing University
3 PUBLICATIONS **2** CITATIONS
SEE PROFILE

International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 35, No. 13 (2021) 2150033 (26 pages)

DOI: 10.1142/S0218001421500336

World Scientific
www.worldscientific.com

Multi-Feature Broad Learning System for Image Classification

Ran Liu*, Yaqiong Liu*, Yang Zhao*, Xi Chen*, Shanshan Cui*,
Feifei Wang* and Lin Yi†,‡

*College of Computer Science, Chongqing University
Chongqing, P. R. China

†Chongqing University Cancer Hospital
Chongqing, P. R. China
‡linyi_cqu@163.com

Received 1 August 2020
Accepted 26 April 2021
Published

A multi-feature broad learning system (MFBLS) is proposed to improve the image classification performance of broad learning system (BLS) and its variants. The model is characterized by two major characteristics: multi-feature extraction method and parallel structure. Multi-feature extraction method is utilized to improve the feature-learning ability of BLS. The method extracts four features of the input image, namely convolutional feature, K-means feature, HOG feature and color feature. Besides, a parallel architecture that is suitable for multi-feature extraction is proposed for MFBLS. There are four feature blocks and one fusion block in this structure. The extracted features are used directly as the feature nodes in the feature block. In addition, a "stacking with ridge regression" strategy is applied to the fusion block to get the final output of MFBLS. Experimental results show that MFBLS achieves the accuracies of 92.25%, 81.03%, and 54.66% on SVHN, CIFAR-10, and CIFAR-100, respectively, which outperforms BLS and its variants. Besides, it is even superior to the deep network, convolutional deep belief network, in both accuracy and training time on CIFAR-10. Code for the paper is available at https://github.com/threedteam/mfbls.

Keywords: Broad learning system; image classification; multi-feature extraction; parallel structure; neural network.

1. Introduction

Image classification is a hot topic in image processing and artificial intelligence.[1,28,36,41] Many studies focus on how to improve the performance of image classification and image recognition.[4,6,36] In recent years, deep learning has been widely used for image classification for its excellent performance.[14,35,47] Various deep

‡Corresponding author.

networks, such as deep belief network (DBN),[15] deep Boltzmann machine (DBM),[34,42] and deep convolutional neural network (DCNN),[22,35] have been developed and tested for image classification. Among these networks, DCNN is more widely used in image classification because of its ability to learn high-level features.[25,40,41] The state-of-the-art DCNNs, such as AlexNet,[22] GoogLeNet,[37] and ResNet,[14] have achieved excellent results in datasets such as MNIST,[23] SVHN,[30] CIFAR-10,[21] CIFAR-100,[21] and ImageNet.[8] However, deep networks also have disadvantages: (1) both the parameters and the layers of the networks are numerous[39]; (2) repeatedly parameters updating is required to get excellent performance[50]; (3) a large number of samples are needed to train the model.[50] All these disadvantages make the training process time-consuming.[18,28]

Chen and Liu proposed a Broad Learning System (BLS) to address these problems.[2] BLS is a random vector single-layer neural network with flat functional-link structure.[28] It is a typical random vector functional link neural network (RVFLNN).[17,31,32] BLS uses the ridge regression approximation of pseudoinverse[2] to calculate the optimal output weights, which needs no iterations; the other weights and biases in BLS are all generated randomly. Therefore, the training time of BLS is surprisingly low. Although BLS achieved excellent classification results on simple datasets such as MNIST[23] and NORB,[24] it fails to learn effectively and efficiently the features from more complex image datasets such as SVHN,[30] CIFAR-10,[21] and CIFAR-100.[21] How to improve the classification performance of BLS for complex image datasets is a challenging task.

In previous research, Liu *et al.*[28] proposed a K-means-BLS which combined K-means clustering and BLS to improve the image classification performance of BLS. Features extracted by K-means are imported directly into BLS for classification. Jin *et al.*[18] proposed a graph regularized broad learning system (GBLS) to improve the classification ability. One useful trick for this improvement is that graph regularization term is introduced into the objective function to constrain the output weights. Yang proposed a CNN-based broad learning system (CNNBLS)[46] in which convolution and max-pooling are used to extract features from images. Zhou and He proposed a broad learning model based on enhanced feature learning.[50] The model adds a hidden layer on enhancement nodes of BLS to better learn the features of input. We refer to this model as EFBLS.

Although these models can do improve the classification performance of BLS, the improvement is still limited, especially for complex image datasets such as CIFAR-10. This is mainly because these models extract only a single feature, hence cannot fully learn the features from the datasets.

In this paper, we propose a so-called multi-feature broad learning system (MFBLS) to solve the problem above. Instead of extracting only a single feature in BLS and its variants, MFBLS extract four carefully selected features, namely convolutional feature, K-means feature, HOG feature, and color feature, to improve the classification performance as well as keep low training time. Besides, a parallel

architecture that is perfectly suitable for multi-feature extraction is proposed for MFBLS. Note that these four features are extracted by algorithms of convolutional feature extraction,[3] K-means clustering,[28] histograms of oriented gradient (HOG),[7] and color feature extraction, respectively. We consider the convolutional feature of images in our model for its advantages such as invariance to shift, distortion and scale, strong robustness, and strong fault tolerance.[36,46] K-means feature is the mapped visual dictionary obtained by K-means clustering from the input image.[28] It can be used as the local feature of the image. HOG feature is a global descriptor of the image and is often used for image classification together with Support Vector Machine (SVM).[7] HOG captures edge information and local intensity gradients; hence, it is also a good texture descriptor.[44] Color is the most intuitive feeling of human eyes. The color feature is not easily disturbed by external factors. Therefore, the color feature is also included in our model. Unlike other variants of BLS, the extracted features in MFBLS are directly used as the mapped features (i.e. feature nodes) of BLS; hence, no feature mapping is needed. As the four extracted features are from different feature spaces, we do not concatenate these features and "enhance" them as a whole, but "enhance" them separately. As a result, it forms a parallel structure. The final implementation indicates that the multi-feature extraction and parallel architecture are beneficial to the performance improvement of BLS. The main contributions of this paper are as follows:

- We propose a new variant of BLS, called multi-feature broad learning system (MFBLS), for efficient image classification on complex data sets (SVHN, CIFAR-10, and CIFAR-100). The model has the advantages of short training time and high classification accuracy.
- A multi-feature extraction method is proposed to significantly improve the learning ability of BLS. Instead of using a single feature in BLS and its variants, the method uses four different features in its feature mapping portion.[3] As different features are different descriptors of the input image, our model can learn more information from the input image. In addition, a convolutional-feature-extraction method without training process is proposed. The method uses only four convolutional layers and four max-pooling layers for feature extraction. Weights in convolutional layers are randomly initialized and then do not be updated anymore.
- A parallel structure suitable for multi-feature extraction is presented. There are four feature blocks (convolutional feature block, K-means feature block, HOG feature block, and color feature block) in this structure, which are used to enhance four types of extracted features, respectively. Each feature block is composed of feature nodes, enhancement nodes, and output nodes. These feature blocks are parallel to each other, which makes the structure easy to extend. In addition, a fusion block is proposed to fuse the output of the four blocks, and a stacking with ridge regression strategy is used in this block for output weights optimization so as to get a better result. Therefore, the parallel structure contains four feature blocks and one fusion block.

The rest of this paper is arranged as follows. In Sec. 2, we briefly review the related work: broad learning system and K-means clustering. We will then address the proposed MFBLS model in Sec. 3 in detail. Experimental results and analysis are followed in Sec. 4. Conclusions can be found in Sec. 5.

2. Preliminaries

2.1. *Broad learning system*

The broad learning system,[2] proposed by Chen and Liu, is a novel network architecture for rapid classification and regression. It is a flat single-layer neural network.[50] First, the BLS performs feature mapping on input data to form feature nodes; second, the feature nodes are enhanced to form the enhancement nodes; finally, the feature nodes and the enhancement nodes are connected to the output nodes,[18] and the desired output weights of the network are calculated by ridge regression of the pseudoinverse.[16]

We suppose the input data $\mathbf{X} \in \mathbb{R}^{N\times d}$ contains N samples, and the dimension of the samples is d. The label set of $\mathbf{X}$ is $\mathbf{Y} \in \mathbb{R}^{N\times c}$, where c is the number of classes. Figure 1 shows the structure of a typical BLS, and the construction of BLS is given in the following.

First, $\mathbf{X}$ is mapped randomly to form the ith group of mapped features $\mathbf{Z}_i$ by feature mapping ϕ_i:

$$\mathbf{Z}_i = \phi_i(\mathbf{X}\mathbf{W}_{ei} + \boldsymbol{\beta}_{ei}), \quad i = 1, \ldots, n, \tag{1}$$

where weights $\mathbf{W}_{ei}$ and biases $\boldsymbol{\beta}_{ei}$ are generated randomly and not updated during the training stage. By mapping $\mathbf{X}$ for n times, n groups of feature nodes can be formed. Let $\mathbf{Z}$ denote the collection of all feature nodes (mapped features):

$$\mathbf{Z} = [\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_n]. \tag{2}$$

Feature mapping ϕ_i can be the same or different for the n-time random mappings.

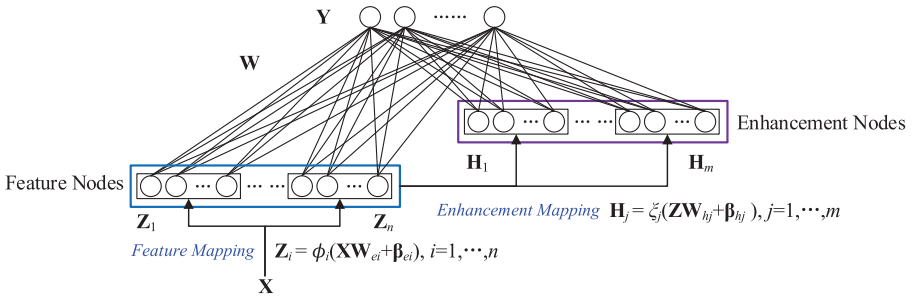


Fig. 1. Structure of a typical BLS.

Second, the collection of all feature nodes $\mathbf{Z}$ is transformed to construct the jth group of enhancement nodes $\mathbf{H}_j$:

$$\mathbf{H}_j = \xi_j(\mathbf{Z}\mathbf{W}_{hj} + \boldsymbol{\beta}_{hj}), \quad j = 1, \ldots, m, \tag{3}$$

where weights $\mathbf{W}_{hj}$ and biases $\boldsymbol{\beta}_{hj}$ are generated randomly. ξ_j is a nonlinear activation function (enhancement mapping).[27] $\mathbf{Z}$ is enhanced for m-times to form m groups of enhancement nodes (enhanced features),[50] denoted as $\mathbf{H}$:

$$\mathbf{H} = [\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_m]. \tag{4}$$

Similarly, function ξ_j can be the same or different for the m-times random mappings.

Then, the mapped features $\mathbf{Z}$ and the enhanced features $\mathbf{H}$ are concatenated, denoted as $\mathbf{A}$:

$$\mathbf{A} = [\mathbf{Z} \mid \mathbf{H}] = [\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_n \mid \mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_m]. \tag{5}$$

Finally, the output $\hat{\mathbf{Y}}$ of the BLS is represented as

$$\hat{\mathbf{Y}} = \mathbf{A}\mathbf{W} = [\mathbf{Z} \mid \mathbf{H}]\mathbf{W}, \tag{6}$$

where $\mathbf{W}$ is the output weights. BLS can obtain the optimal $\mathbf{W}$ by minimizing the following objective function:[18,27]

$$\arg\min_{\mathbf{W}} : \|\mathbf{Y} - \mathbf{A}\mathbf{W}\|_2^2 + \lambda\|\mathbf{W}\|_2^2, \tag{7}$$

where the first term represents the error (training error)[18] between the true value and the output value of BLS, and the second term is the regularization term,[18] which controls the complexity of BLS. Parameter λ is the regularization parameter. The ridge regression method is used to solve Eq. (7), and $\mathbf{W}$ can be approximated by

$$\mathbf{W} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{Y}, \tag{8}$$

where $\mathbf{I}$ is the identity matrix.

2.2. *K-means clustering*

K-means clustering is widely used for feature learning and representation of images.[41] A typical K-means clustering algorithm finds k cluster centroids that minimize the distance between the input samples and the nearest cluster centroid.[5] K-means clustering can be regarded as a procedure of learning a dictionary $\mathbf{D} \in \mathbb{R}^{d\times k}$ which is composed of k cluster centroids, where d is the dimension of the sample vector. Reference 5 uses an improved version of K-means clustering ("gain shape" vector quantization[48] or "spherical K-means"[10]) to obtain dictionary $\mathbf{D}$:

$$\begin{aligned} &\arg\min_{\mathbf{D},\boldsymbol{s}} \sum_i \|\mathbf{D}\boldsymbol{s}^{(i)} - \boldsymbol{x}^{(i)}\|_2^2 \\ &\text{subject to } \|\boldsymbol{s}^{(i)}\|_0 \le 1, \quad \forall i \in 1, \ldots, N \\ &\text{and } \|\mathbf{D}^{(j)}\|_2 = 1, \quad \forall j \in 1, \ldots, k, \end{aligned} \tag{9}$$

where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is the ith input sample, $\boldsymbol{s}^{(i)} \in \mathbb{R}^k$ is the corresponding code vector, and N is the number of samples. $\mathbf{D}^{(j)}$ is the jth cluster centroid. It can be known from (9) that the optimal $\mathbf{D}$ and code vector $\boldsymbol{s}^{(i)}$ are found to minimize the reconstruction error. The reconstruction error is measured by the squared difference between $\boldsymbol{x}^{(i)}$ and $\mathbf{D}\boldsymbol{s}^{(i)}$. However, this optimization is constrained. Expression $\|\boldsymbol{s}^{(i)}\|_0 \leq 1$ constrains each code vector to have at most one nonzero term, which ensures a very simple new representation of $\boldsymbol{x}^{(i)}$. The expression $\|\mathbf{D}^{(j)}\|_2 = 1$ constrains each cluster centroid to be a unit vector. This prevents $\mathbf{D}^{(j)}$ from becoming arbitrarily large or small.

K-means clustering algorithm solves (9) by alternating optimization of $\mathbf{D}$ and $\boldsymbol{s}$.[5] Given $\mathbf{D}$, the following formula is used to solve the optimal $\boldsymbol{s}^{(i)}$:

$$\boldsymbol{s}_j^{(i)} = \begin{cases} \mathbf{D}^{(j)T}\boldsymbol{x}^{(i)}, & \text{if } j == \arg\max_l |\mathbf{D}^{(l)T}\boldsymbol{x}^{(i)}| \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Hence, an optimal $\mathbf{D}$ can be obtained[5]:

$$\begin{aligned} &\mathbf{D} := \mathbf{X}_K\mathbf{S}_K^T + \mathbf{D} \\ &\mathbf{D}^{(j)} := \mathbf{D}^{(j)}/\|\mathbf{D}^{(j)}\|_2, \quad \forall\, j \in 1, \ldots, k, \end{aligned} \tag{11}$$

where $\mathbf{X}_K \in \mathbb{R}^{d\times N}$ is a matrix whose columns are the input samples $\boldsymbol{x}^{(i)}(i \in [1, N])$. $\mathbf{S}_K \in \mathbb{R}^{k\times N}$ is the matrix whose columns are the code vectors $\boldsymbol{s}^{(i)}$ from (10). Equations (10) and (11) are relatively simple, hence K-means clustering can learn a large dictionary quickly.[5] At the same time, learning the K-means dictionary does not require other parameters except k, which makes learning easy.

Note that the inputs of the K-means clustering algorithm are images (or image patches) in our study. These image patches are preprocessed to get $\boldsymbol{x}^{(i)}$ by normalization and whitening.[5] Detailed steps of the K-means clustering algorithm are as follows:

(1) Input normalization:

$$\boldsymbol{x}^{(i)} = \frac{\tilde{\boldsymbol{x}}^{(i)} - \text{mean}(\tilde{\boldsymbol{x}}^{(i)})}{\sqrt{\text{var}(\tilde{\boldsymbol{x}}^{(i)}) + \varepsilon_{\text{norm}}}}, \quad \forall\, i \in 1, \ldots, N, \tag{12}$$

where $\tilde{\boldsymbol{x}}^{(i)}$ is an original image or image patch and "mean" and "var" are the mean and variance of the elements of $\tilde{\boldsymbol{x}}^{(i)}$, respectively. $\boldsymbol{x}^{(i)}$ is the normalized sample. $\varepsilon_{\text{norm}}$ is a small constant which is used to avoid dividing by zero as well as suppress noise.

(2) ZCA whitening:

$$\begin{aligned} &[V, E] = \text{eig}(\text{cov}(\mathbf{X}_K)) \\ &\boldsymbol{x}^{(i)} = V(E + \mathcal{E}_{zca}\mathbf{I})^{-1/2}V^T\boldsymbol{x}^{(i)}, \quad \forall\, i \in 1, \ldots, N, \end{aligned} \tag{13}$$

where "cov" is the covariance of inputs, "eig" is the eigenvalue decomposition of inputs, V denotes the feature vector, E denotes the eigenvalue, and ε_{zca} is a small constant parameter. ZCA whitening can eliminate the correlation between learned cluster centroids.[5]

(3) Dictionary learning:
We use (10) and (11) to optimize $\boldsymbol{s}^{(i)}$ and $\mathbf{D}$ alternately until convergence. Usually, 50 iterations are enough in our study.

3. The Proposed Method

To improve the image classification ability of BLS, we propose a multi-feature broad learning system, abbreviated as MFBLS. The key point of the model is that multi-feature extraction is used to replace the feature mapping in BLS. As multiple features are extracted from different feature spaces in our model, we need to utilize these features in a proper way so as to be able to get optimal output.

It is fairly easy to think of concatenating all extracted features together and making the concatenated result as the feature nodes directly in the BLS. Then we can get an MFBLS with concatenated structure, as shown in Fig. 2.

However, there are problems for the concatenated structure: concatenating features from different feature spaces directly may lead to the loss of feature information, which will weaken the classification performance. Besides, the dimension of the concatenated feature tends to become too high, which will result in a time-consuming calculation of output weights.

To solve the above problems, a parallel structure suitable for multi-feature extraction is proposed for MFBLS, as shown in Fig. 3. The structure contains four feature blocks (convolutional feature block, K-means feature block, HOG feature block, and color feature block), and a fusion block. The four feature blocks take the convolutional feature, K-means feature, HOG feature, and color feature as input, respectively. Each feature block uses its input directly as the feature nodes and then enhances the feature nodes by using the enhancement mapping ξ_* ($*$ can be F, K, H, and S). F corresponds to the convolutional feature, K to the K-means feature, H to the HOG feature, and S to the color feature. Feature nodes and enhancement nodes are connected to output nodes. Each feature block uses the ridge regression approximation of pseudoinverse to solve its output weights $\mathbf{W}_*$ ($*$ can be F, K, H, and S). Note the output should be normalized. The normalized outputs of the four feature

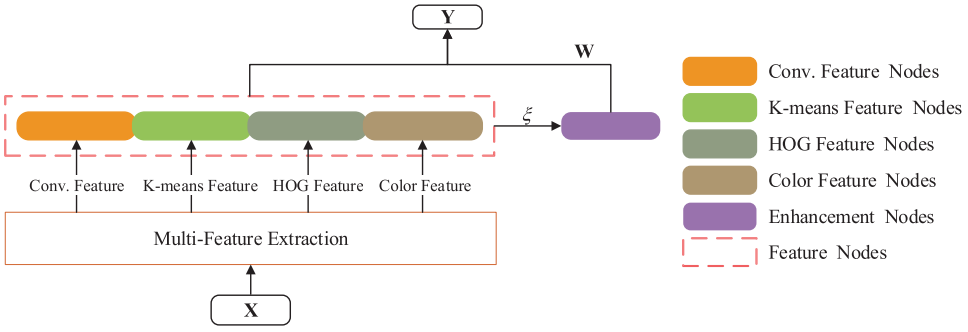


Fig. 2. MFBLS with concatenated structure. Note that the type of features can be extended as needed in this structure.

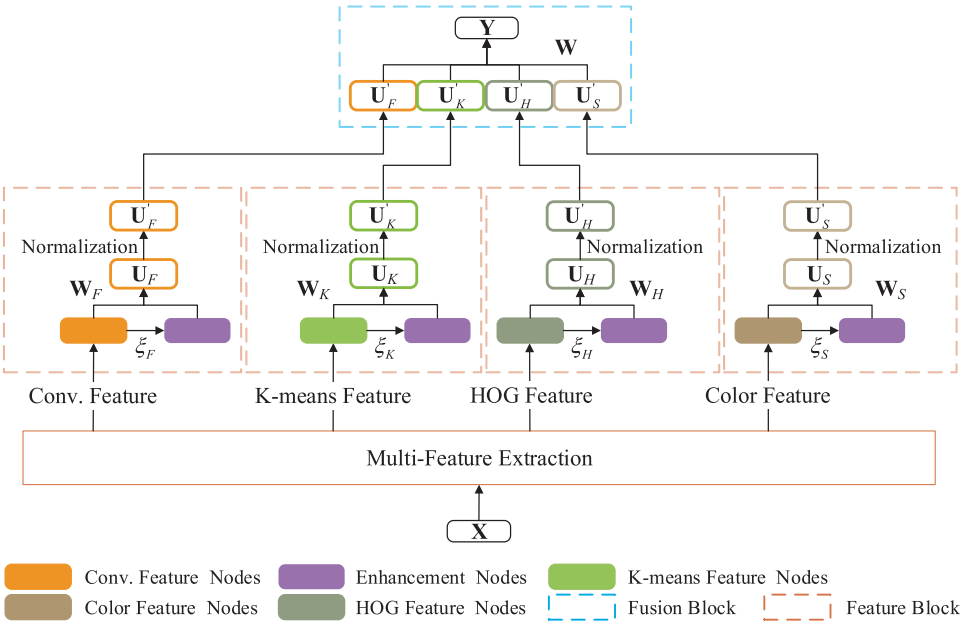


Fig. 3. Structure of MFBLS. It is constructed using a parallel structure, in which multiple feature blocks and one fusion block are contained.

blocks are $\mathbf{U}'_F$, $\mathbf{U}'_K$, $\mathbf{U}'_H$, and $\mathbf{U}'_S$, respectively. They are inputted to the fusion block. The fusion block concatenates them and uses the "stacking with ridge regression" strategy to obtain the final prediction of MFBLS.

In summary, the multi-feature extraction method and the parallel structure are two important aspects of our MFBLS model. The details of each aspect are addressed in the following sections.

3.1. *Multi-feature extraction*

In the original BLS, input data are mapped randomly to form mapped features. However, this random mapping may not learn image features effectively if the input data are images. Therefore, we put forward a multi-feature extraction method instead of random mapping for image feature learning. Four features, convolutional feature, K-means feature, HOG feature, and color feature, are extracted in MFBLS. In this section, we describe the multi-feature extraction in detail. It should be noted that each feature extraction procedure is independent of others.

3.1.1. *Convolutional feature extraction*

MFBLS extracts the convolutional feature and uses it as the mapped feature. Convolution and max-pooling are used to extract the features of the image. Unlike DCNN, the weights in convolutional layers in MFBLS are not updated after initialization.

Therefore, training is not required for feature extraction, which reduces the training time of MFBLS greatly. What is more, different from CNNBLS, we only keep the results of the last max-pooling layer and use principal component analysis (PCA)[11] to reduce the dimension of the feature. The procedure of convolutional feature extraction is shown in Fig. 4.

(1) The input image is first convolved and then pooled. These two operations are performed alternately for four times, so there are four convolutional layers (C1–C4 in Fig. 4) and four max-pooling layers (P1–P4 in Fig. 4) in our convolutional feature extraction network.
(2) The result obtained after the last pooling operation is flattened into a vector.
(3) PCA is applied to the flattened vector. The final result is the convolutional feature.

As for the convolutional layers, the size of the convolution kernels in C1, C2, and C3 is 3×3; while in C4, the kernel size equals the size of the input of C4. The padding mode for all the convolutional layers is zero padding, and the stride is 1. The number of convolution kernels of each layer is set to

$$\frac{N_{\text{Train}} \times \gamma}{h_{\text{Input}} \times w_{\text{Input}}}, \tag{14}$$

where N_{Train} is the number of training samples. The height and width of the input for the current layer are h_{Input} and w_{Input}, respectively. The hyperparameter that needs to be set is γ. Different values of γ can be applied for different datasets. Kernels are randomly generated with a normal distribution. Each pooling layer is max-pooling layer with 2×2 size, 2 stride, and nonpadding. The purpose of max-pooling is to reduce the dimension of the inputs. Besides, unlike CNNBLS, MFBLS only uses the output of P4, which can avoid the high dimension of convolutional feature and reduce computational complexity.

3.1.2. *K-means feature extraction*

K-means clustering is an efficient unsupervised clustering algorithm. It can be used to extract local features from images. K-means feature extraction from input images in MFBLS can be divided into two stages, namely the learning stage and the representation stage:

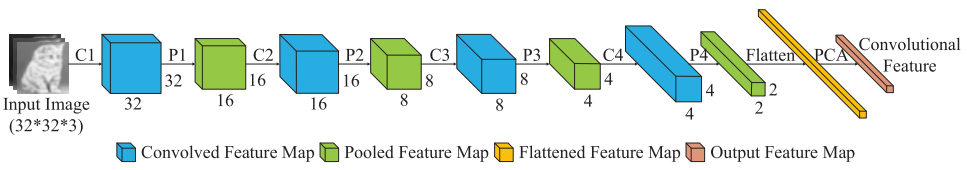


Fig. 4. Flowchart of convolutional feature extraction.

In the learning stage, the dictionary $\mathbf{D}$ is obtained. First, image patches are sampled randomly from input images and preprocessed by normalization and ZCA whitening. Then, dictionary learning is carried out on these patches to obtain $\mathbf{D}$. See Sec. 2.2 for details.

In the representation stage, we use dictionary $\mathbf{D}$ to generate a feature vector for image representation. As can be seen from Fig. 5, a 6×6 window slides through the three-channel color image with stride 1 for sampling. Note the window size needs to be the same size as the size of the image patch in the learning stage. After sampling, many image patches can be obtained, and each patch is a multi-dimensional vector, denoted as $\boldsymbol{x} \in \mathbb{R}^d$. We adopt the mapping function $f : \mathbb{R}^d \to \mathbb{R}^k$ to transform each image patch into a feature vector:

$$\begin{aligned} &f(\boldsymbol{x}; \mathbf{D}) = (\max\{0, \bar{d} - d_1\}, \ldots, \max\{0, \bar{d} - d_j\} \ldots, \max\{0, \bar{d} - d_k\}) \\ &d_j = \|\boldsymbol{x} - \mathbf{D}^{(j)}\|_2, \quad \forall j \in 1, \ldots, k \\ &\bar{d} = \frac{1}{k} \sum_{j=1}^{k} d_j, \end{aligned} \tag{15}$$

where $\mathbf{D}^{(j)}$ is the jth cluster centroid, d_j is the distance between image patch $\boldsymbol{x}$ and the jth cluster centroid, and k is the number of cluster centroids. After mapping, each image patch is converted to a k-dimensional vector. Then we use sum-pooling for dimension reduction: all the image patches are divided into four parts, and sum-pooling is performed for each part. Finally, the sum-pooling results are concatenated and standardized to be a feature vector whose dimension is $4k$. The extracted K-means feature replaces the mapped feature in BLS.

3.1.3. *HOG feature extraction*

HOG feature describes the gradients of the image local area in different directions. It is a common descriptor describing image texture features, which is often used for object detection.[7,44] Hence, the HOG feature is introduced in our model. The procedure of HOG feature extraction is as follows[7]:

(1) Convert the input color image to grayscale image and normalize the grayscale image to reduce the influence of illumination variation.[7]

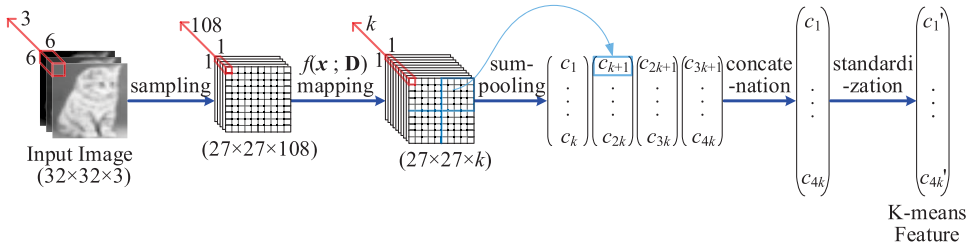


Fig. 5. Flowchart of the representation stage for K-means feature extraction with dictionary $\mathbf{D}$. Take the 32×32 color input image as an example.

(2) Calculate the gradient of each pixel in the image.
(3) The normalized image is segmented into small image patches called cells. A 1-D gradient histogram vector for each cell is computed by accumulating the gradient information of each pixel in the cell.
(4) Multiple cells form a larger block. Then normalize all the vectors within the block. This allows the extracted feature to have better invariance to the illumination.
(5) HOG feature is obtained by combining the histograms of all blocks into one feature vector. The HOG feature extracted acts as a mapped feature in BLS.

3.1.4. *Color feature extraction*

The color feature is the most intuitive feature of an image. It is a global descriptor. The color feature is usually represented by color histogram,[38] color moment,[33] color correlogram,[33] etc. In this paper, we choose color histogram and color moment to represent the color feature of an image. The procedure of color feature extraction is as follows:

(1) Transform the image from RGB color space to HSV color space. The reason why the color feature is extracted in HSV color space is that the HSV color space provides an intuitive representation of color for human visual perception.[29]
(2) Use 6, 4, and 4 bins to calculate the color histogram respectively for each channel of the image in HSV color space. Consequently, a 96-dimensional color histogram vector is obtained.
(3) Calculate three color moments (mean, standard deviation, and skewness) for each of the three channels of the image. These three color moments are concatenated into one nine-dimensional color moment vector.
(4) Color histogram vector and color moment vector are concatenated into a 105-dimensional color feature vector. This color feature is used as the mapped feature in BLS.

3.2. *Parallel structure*

After obtaining multiple features of the input images, MFBLS performs classification based on these features. In order to improve the classification performance as well as make the model easy to extend, we propose a parallel structure for MFBLS. As mentioned in previous sections, there are four feature blocks and one fusion block in this structure. Different from the structure of BLS, our parallel structure uses extracted features directly as the feature nodes in feature block; hence, no random feature mapping is needed for feature nodes generation. In addition, the fusion block uses the "stacking with ridge regression" strategy to get the final output of MFBLS. The details of the parallel structure are addressed in the following sections.

3.2.1. *Feature block*

The structure of the four feature blocks (convolutional feature block, K-means feature block, HOG feature block, and color feature block) is similar, so their

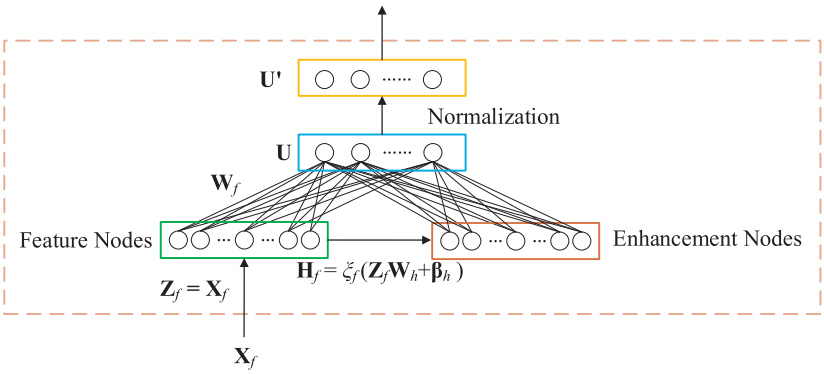


Fig. 6. Structure of a typical feature block in MFBLS.

construction process is also similar. A typical feature block is shown in Fig. 6. The input of the feature block is one type of the features extracted, and the output is a matrix (a refined feature) with classification information obtained from the input. The establishment of a typical feature block is described in the following.

First, the feature nodes are constructed. Assume that the input of MFBLS is an image dataset $\mathbf{X}$, which contains N samples. One type of feature set extracted from $\mathbf{X}$ can be denoted as $\mathbf{X}_f \in \mathbb{R}^{N\times d_f}$, where d_f is the dimension of this type of feature. That is, the feature of each sample is a d_f-dimensional vector $\boldsymbol{x}_f$. Then the input of the feature block is

$$\mathbf{X}_f = [\boldsymbol{x}_{f1}, \boldsymbol{x}_{f2}, \ldots, \boldsymbol{x}_{fN}]^T. \tag{16}$$

We use $\mathbf{X}_f$ to build the feature nodes $\mathbf{Z}_f$ directly:

$$\mathbf{Z}_f = \mathbf{X}_f. \tag{17}$$

That means the feature nodes of BLS are replaced with the extracted features directly in our feature block. It should be noted that there are multiple groups of feature nodes in BLS[3]; however, there is only one group of feature nodes in the feature block.

Second, the enhancement nodes of the feature block are obtained by

$$\mathbf{H}_f = \xi_f(\mathbf{Z}_f\mathbf{W}_h + \boldsymbol{\beta}_h), \tag{18}$$

where the weights $\mathbf{W}_h$ and biases $\boldsymbol{\beta}_h$ are randomly generated, and ξ_f is a nonlinear activation function. Different from the feature nodes in BLS which are enhanced multiple times, the feature nodes in feature block only enhanced once; hence, only one group of enhancement nodes is generated.

Third, the output nodes can be obtained by

$$\mathbf{U} = [\mathbf{Z}_f \,|\, \mathbf{H}_f]\mathbf{W}_f = \mathbf{A}_f\mathbf{W}_f, \tag{19}$$

where $\mathbf{A}_f = [\mathbf{Z}_f \,|\, \mathbf{H}_f]$, $\mathbf{W}_f$ is the output weights of the feature block. $\mathbf{W}_f$ can be calculated by

$$\mathbf{W}_f = (\mathbf{A}_f^T \mathbf{A}_f + \lambda_f \mathbf{I})^{-1} \mathbf{A}_f^T \mathbf{Y}, \tag{20}$$

where $\mathbf{Y}$ is the label set, and λ_f is the regularization parameter.

Finally, we normalize $\mathbf{U}$ to $\mathbf{U}'$ by using softmax function.[12] Suppose $\boldsymbol{u}_i (i \in [1, N])$ is the prediction vector of sample i, then each component of $\boldsymbol{u}_i$ is normalized to

$$u'_{ij} = \frac{e^{u_{ij}}}{\sum_{j=1}^{c} e^{u_{ij}}}, \tag{21}$$

where u_{ij} is the probability that sample i belongs to class j, and c is the number of classes. The output $\mathbf{U}'$ is a matrix whose rows are $\boldsymbol{u}_i$.

3.2.2. *Fusion block*

The four feature blocks output four matrices to the fusion block, we call them $\mathbf{U}'_F$, $\mathbf{U}'_K$, $\mathbf{U}'_H$, and $\mathbf{U}'_S$, where the subscript F corresponds to the convolutional feature, K to the K-means feature, H to the HOG feature, and S to the color feature. The fusion block needs to fuse these outputs to produce the final output of MFBLS. We propose a fusion strategy, called stacking with ridge regression, to fuse the outputs. This strategy treats the four types of outputs as features and serves them as the feature nodes of BLS, and then uses the ridge regression method to solve the optimal output weights, and finally obtains the final output of MFBLS. The structure of the fusion block is shown in Fig. 7. It is similar to the structure of the feature block. For simplicity, we do not use the enhancement nodes in fusion block.

As can be seen from Fig. 7, the feature nodes in fusion block are

$$\mathbf{Z} = [\mathbf{U}'_F, \mathbf{U}'_K, \mathbf{U}'_H, \mathbf{U}'_S]. \tag{22}$$

Therefore, the output $\hat{\mathbf{Y}}$ of the fusion block has the following form:

$$\hat{\mathbf{Y}} = \mathbf{Z}\mathbf{W} = \mathbf{A}\mathbf{W}, \tag{23}$$

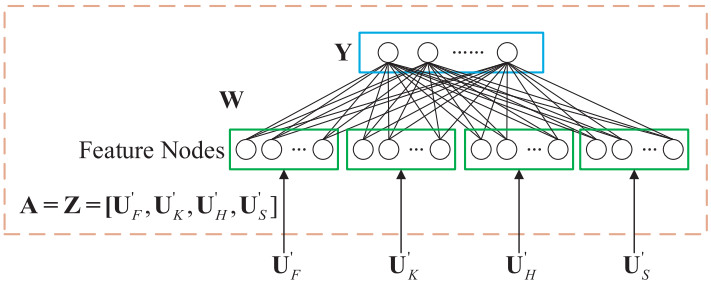


Fig. 7. Structure of the fusion block in MFBLS.

where $\mathbf{A} = \mathbf{Z}$. An optimal $\mathbf{W}$ can be calculated by using (7) and (8); thus, we can obtain the $\hat{\mathbf{Y}}$:

$$\hat{\mathbf{Y}} = \mathbf{A}\mathbf{W} = \mathbf{A}\big(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I}\big)^{-1}\mathbf{A}^T\mathbf{Y}, \tag{24}$$

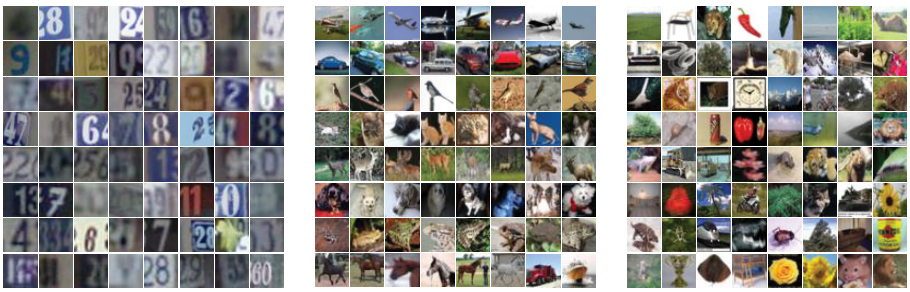
where λ is the regularization parameter. It should be noted that $\hat{\mathbf{Y}}$ is the output of the fusion block, as well as that of MFBLS.

4. Experiment and Discussion

We conduct experiments on SVHN, CIFAR-10, and CIFAR-100 datasets to verify the performance of MFBLS. We implement the model in Python in Ubuntu operating system and perform all of the experiments using an Intel Xeon E5-2678 CPU, an NVIDIA TITAN Xp GPU, and 128G memory. In this section, we first compare MFBLS to other state-of-the-art models. Then, the superiority of the proposed parallel structure for MFBLS is demonstrated by comparing it with a "concatenated structure". Next, the proposed fusion strategy is compared with other fusion strategies. Finally, we conduct parameter sensitivity analysis to illustrate the influence of hyperparameters on the classification performance of MFBLS.

4.1. *Datasets and setting*

Dataset SVHN[30] is similar to MNIST,[23] both of which are digit databases.[30] However, the background of the images in SVHN is more complex. Therefore, classifying images in SVHN is more challenging than that in MNIST. SVHN consists of a training set, an extra set, and a test set. The training set contains 73 257 samples, the extra set has 531 131 samples, and the test set owns 26 032 samples. There are 10 classes in SVHN. Some images from SVHN are shown in Fig. 8(a). Some researchers used training set and extra set together as a new training set.[9,13,26] However, the new training set obtained in this way are too large, nearly 600k, which will increase the training time and memory cost greatly. To overcome this problem, we select 36 743 samples randomly from the extra set. These samples are added to the training set to



(a) (b) (c)

Fig. 8. Samples from three datasets: (a) SVHN, (b) CIFAR-10, and (c) CIFAR-100.

form a new training set with 11 000 samples. In addition, we select 1000 samples from the remaining extra set to form a validation set. The details are shown in Table 1.

CIFAR-10[21] consists of 60 000 RGB images with a resolution of 32×32, in which 50 000 are training samples and 10 000 are test samples. There are also 10 classes in this dataset. Each class contains 5000 training images and 1000 test images. These classes include planes, cars, birds, cats, deer, dogs, frogs, horses, boats, and trucks. The background and objects in these images are more complex than those in images from SVHN. Some images from CIFAR-10 are shown in Fig. 8(b). In our experiments, we select 1000 samples randomly from the training set to form a validation set. The details are shown in Table 1.

CIFAR-100[21] is similar to CIFAR-10 in that the images in it are also RGB images with a resolution of 32×32. The difference is that the classification of CIFAR-100 is more elaborate. The dataset consists of 100 classes. There are 500 training samples and 100 test samples in each class. These 100 classes are further divided into 20 superclasses. For example, the flower superclass contains five classes: orchid, poppy, rose, sunflower, and tulip. This increases the difficulty of learning greatly. Some images from CIFAR-100 are shown in Fig. 8(c). We select 1000 images randomly from the training set to form a validation set in our experiments. The details are shown in Table 1.

The hyperparameters for multi-feature extraction in MFBLS include (1) the size of the image patch and the number of cluster centroids k when we extract the K-means feature. For SVHN, the size of the image patch is 8×8, and k is 500; for CIFAR-10 or CIFAR-100, the size of the image patch is 6×6, and k is 1024 and 1300, respectively. It should be noted that the K-means feature is extracted from the grayscale images for SVHN; while for CIFAR-10 and CIFAR-100, it is extracted from the three channels of the RGB images; (2) the hyperparameter γ used to calculate the number of convolution kernels and PCA hyperparameter p when we extract the convolutional feature. For SVHN, CIFAR-10, and CIFAR-100, the values of γ are 0.2, 0.18, and 0.18, respectively. For all datasets, p is 0.99.

The relevant hyperparameters of MFBLS model include (1) the regularization parameters λ_H, λ_S, λ_K, and λ_F for feature blocks, and the regularization parameter λ for fusion block; (2) the scaling parameters S_H, S_S, S_K, and S_F for generating enhancement nodes; (3) the numbers of the enhancement nodes E_H, E_S, E_K, and E_F. The hyperparameter settings for each dataset are shown in Table 2. The influence of these hyperparameters on the classification results will be analyzed in detail in Sec. 4.5.

Table 1. Information on the datasets for experiments.

Dataset	Train	Validation	Test	Classes
SVHN	110 000	1000	26 032	10
CIFAR-10	49 000	1000	10 000	10
CIFAR-100	49 000	1000	10 000	100

Table 2. Hyperparameter settings of MFBLS for different datasets.

	Conv.			K-Means			HOG			Color			Fusion
Dataset	λ_F	S_F	E_F	λ_K	S_K	E_K	λ_H	S_H	E_H	λ_S	S_S	E_S	λ
SVHN	0.010	0.80	5000	0.005	0.90	9900	0.002	0.65	6500	0.500	0.70	1000	1.000
CIFAR-10	0.001	0.85	10 000	0.001	0.70	10 000	0.005	0.70	10 000	0.001	0.85	5000	1.000
CIFAR-100	0.010	0.85	9900	0.01	0.95	9900	0.001	0.71	9500	0.001	0.95	9500	0.005

4.2. *Performance evaluation*

We compare the proposed model with other state-of-the-art models on SVHN, CIFAR-10, and CIFAR-100 for classification performance evaluation. The compared models include BLS,[2] K-means-BLS,[28] CNNBLS,[46] EFBLS[50] (we refer to the model proposed in Ref. 50 as EFBLS), convolutional DBN,[20] LeNet5,[23] SDT-ELM,[49] and ELM-ARF.[43] The first four models are all broad learning models.[2] Convolutional DBN and LeNet5 are deep networks. SDT-ELM and ELM-ARF are extremely learning machine (ELM)-based networks. Table 3 shows the best test accuracies achieved on the three datasets using MFBLS and other models. From Table 3, we can draw the following conclusions:

(1) Compared with the broad learning models, MFBLS achieves the highest test accuracy of classification for these three datasets. For SVHN, the accuracy of MFBLS is 92.25%, which is 14.6 pp (percentage points), 4.7 pp, 11.0 pp, and 13.9 pp higher than that of BLS, K-means-BLS, CNNBLS, and EFBLS, respectively. For CIFAR-10, MFBLS achieves an accuracy of 81.03%, which is 33.4 pp, 5.0 pp, 20.9 pp, and 28.7 pp higher than that of BLS, K-means BLS, CNNBLS, and EFBLS, respectively. For CIFAR-100, the accuracy of MFBLS is 31.8 pp, 5.0 pp, 12.7 pp, 25.5 pp, and 31.5 pp higher than that of BLS, K-means-BLS, CNNBLS, and EFBLS, respectively. These results demonstrate that the proposed MFBLS improve the learning ability of BLS and its variants significantly on various datasets, especially on complex datasets.

Table 3. Accuracies (%) of MFBLS and the other state-of-the-art methods for SVHN-test, CIFAR-10-test, and CIFAR-100-test.

Method	SVHN	CIFAR-10	CIFAR-100
BLS	77.65	47.62	22.84
K-means-BLS	87.55	76.05	41.95
CNNBLS	81.26	60.12	29.13
EFBLS	78.33	52.35	23.11
Conv. DBN	—	78.90	—
LeNet5	89.46	61.22	30.51
SDT-ELM	82.67	54.89	21.86
ELM-ARF	78.13	60.14	24.74
MFBLS	**92.25**	**81.03**	**54.66**

(2) The classification performance of MFBLS is better than convolutional DBN and LeNet5. For example, on CIFAR-10, the accuracy of MFBLS is 2.1 pp and 19.8 pp higher than that of convolutional DBN and LeNet5, respectively.
(3) MFBLS is superior to SDT-ELM and ELM-ARF. For example, on CIFAR-100, the accuracy of MFBLS is 32.8 pp and 29.9 pp higher than that of SDT-ELM and ELM-ARF, respectively.
(4) For all the datasets, the test accuracy of MFBLS is much higher than that of K-means-BLS. We can conclude that besides the K-means feature, using other features (convolutional feature, HOG feature, and color feature) and parallel structure can significantly improve classification performance.

We list the training time of the MFBLS and other models for the three datasets in Table 4. From Table 4, we have the following observations:

(1) The training time of MFBLS is longer than that of the broad learning models, except for K-means-BLS, but within the acceptable range. The reason for the long training time of K-means-BLS and MFBLS is that extracting K-means feature of images is time-consuming.
(2) The training time of convolutional DBN on CIFAR-10 is 36 h (with NVIDIA GTX 280 GPU),[20] far exceeding the training time of MFBLS. MFBLS outperforms the convolutional DBN both in accuracy and training time.[20] The training time of LeNet5 is longer than that of MFBLS on SVHN, but it is shorter on CIFAR-10 and CIFAR-100.
(3) For CIFAR-10 and CIFAR-100, the training time of MFBLS is longer than that of SDT-ELM and ELM-ARF. For SVHN, the training time of SDT-ELM is longest, followed by that of MFBLS, and finally the training time of ELM-ARF.

In general, the classification accuracy of MFBLS is better than that of the compared method. However, as the feature extraction is time-consuming, the training time of our model is slightly longer than that of some broad learning models, and ELM-based networks. Note that the training time of our model is shorter than that of convolutional DBN.

Table 4. Training time of MFBLS and the other state-of-the-art models for SVHN, CIFAR-10, and CIFAR-100.

Method	SVHN	CIFAR-10	CIFAR-100
BLS	384 s	20 s	181 s
K-means-BLS	984 s	1282 s	1349 s
CNNBLS	373 s	113 s	115 s
EFBLS	464 s	108 s	304 s
Conv. DBN	—	36 h	—
LeNet5	1694 s	626 s	687 s
SDT-ELM	1878 s	882 s	1004 s
ELM-ARF	1397 s	247 s	390 s
MFBLS	1542 s	1137 s	1367 s

Compared with the results of DCNNs,[6] the accuracy of MFBLS is lower, but the training time of MFBLS is significantly shorter. This is because DCNNs have a large number of layers and require more time to train the weights of layers. Another advantage of MFBLS is that no graphics processing unit (GPU) is needed.

4.3. *Structure evaluation*

In this section, we demonstrate that the proposed parallel structure is more suitable for MFBLS than concatenated structures for the three datasets. To ensure fairness of the comparisons, we make the number of feature nodes and the number of enhancement nodes in concatenated structure equal to the sum of the feature nodes and the sum of the enhancement nodes in the four feature blocks in parallel structure, respectively. The regularization and scaling parameters in the concatenated structure are fine-tuned to obtain optimal performance. Experimental results are shown in Table 5.

We can see from Table 5 that the accuracy of MFBLS with parallel structure is better than of MFBLS with concatenated structure for all three datasets. This may be because the features merged cannot represent the image effectively. As the four extracted features are from different spaces, each feature may lose some information if we concatenate them directly, resulting in lower accuracy. On the contrary, if we adopt a parallel structure, each feature is enhanced independently, may retain more information for classification.

It can be also seen from Table 5 that the training time of the parallel structure is less than that of the concatenated structure for all the datasets. We will briefly analyze the time complexity of these two structures. But before the analysis, we need to figure out the time complexity of a typical BLS. As for BLS, the most time-consuming operation is the calculation of pseudoinverse of the system equation.[3] As most of the operation is multiplication, we can only consider the multiplication operation here. The number of multiplication operations needed to solve the pseudoinverse of BLS is

$$\begin{aligned} T &= f(N_{\text{Fea}} + N_{En}) \\ &\approx c_1(N_{\text{Fea}} + N_{En})^3 + c_2(N_{\text{Fea}} + N_{En})^2 + c_3(N_{\text{Fea}} + N_{En}) + c_4, \end{aligned} \tag{25}$$

where N_{Fea} and N_{En} is the number of feature nodes and enhancement nodes, respectively; c_1, c_2, c_3, and c_4 are positive constant coefficients. $T = f(N_{\text{Fea}} + N_{En})$

Table 5. Performance comparison of MFBLS with concatenated structure and MFBLS with parallel structure on SVHN, CIFAR-10, and CIFAR-100.

	MFBLS with Concatenated Structure		MFBLS with Parallel Structure	
Dataset	Accuracy (%)	Training Time (s)	Accuracy (%)	Training Time (s)
SVHN	89.90	1889	**92.25**	**1542**
CIFAR-10	79.60	1868	**81.03**	**1137**
CIFAR-100	54.09	2550	**54.66**	**1367**

represents the number of operations when the sum of the number of feature nodes and enhancement nodes is $N_{\text{Fea}} + N_{En}$. T increases as the input increases.

Consequently, the number of operations needed to solve the pseudoinverse of MFBLS with concatenated structure can be calculated by

$$T_c = f(N_{\text{Fea\_}F} + N_{\text{Fea\_}K} + N_{\text{Fea\_}H} + N_{\text{Fea\_}S} + N_{En\_F} + N_{En\_K} + N_{En\_H} + N_{En\_S}), \tag{26}$$

where $N_{\text{Fea\_}*}$ and N_{En_*} ($*$ can be F, K, H, and S) is the number of feature nodes and enhancement nodes of the same type of feature, respectively.

Similarly, the number of operations needed to solve the pseudoinverse of MFBLS with parallel structure can be calculated by

$$\begin{aligned} T_p = {} & f(N_{\text{Fea\_}F} + N_{En\_F}) + f(N_{\text{Fea\_}K} + N_{En\_K}) + f(N_{\text{Fea\_}H} + N_{En\_H}) \\ & + f(N_{\text{Fea\_}S} + N_{En\_S}) + f(4 \cdot c), \end{aligned} \tag{27}$$

where c is the number of classes. Since c is much smaller than the number of feature nodes and enhancement nodes, $f(4 \cdot c)$ can be ignored. Moreover, we can deduce that $T_p < T_c$. Therefore, the training time of the parallel structure is shorter than that of the concatenated structure.

In summary, the parallel structure is superior to the concatenated structure in both accuracy and training time, indicating that the model with parallel structure is more effective and efficient.

4.4. *Fusion strategies evaluation*

As mentioned in previous sections, the fusion block uses a proposed fusion strategy, stacking with ridge regression, to fuse the outputs of the four parallel feature blocks into a single result. To verify the effectiveness of the proposed fusion strategy for our parallel structure, we compare the strategy with the averaging[45] and voting[19] strategies. The experimental results are shown in Table 6.

As can be seen from Table 6, the accuracy of the proposed strategy on SVHN is 92.25%, which is 1.4 pp and 6.3 pp higher than that of averaging and voting strategies, respectively. The results of the proposed strategy on CIFAR-10 and CIFAR-100 are also better than the average and voting strategy. Our fusion strategy is inspired by the idea of stacking, that is, the outputs of four feature blocks are

Table 6. Performance comparison of different fusion strategies for fusion block in MFBLS on SVHN, CIFAR-10, and CIFAR-100.

	Accuracy (%)		
Dataset	Averaging	Voting	Proposed
SVHN	90.85	86.00	**92.25**
CIFAR-10	78.55	71.17	**81.03**
CIFAR-100	52.76	41.58	**54.66**

regarded as features, and these features are used to learn again. When calculating the final output of some category, MFBLS will comprehensively consider the outputs of all blocks, which is not taken into account by the averaging method and the voting method. Therefore, our model can achieve better performance.

4.5. *Parameter sensitivity*

In this section, parameter sensitivity analysis is conducted to show that the proposed MFBLS can achieve excellent results over a wide range of parameter values. Referring to the hyperparameter settings in Refs. 27, 46 and 50, we select the values of the hyperparameters in our experiments from the following sets:

(1) Regularization parameters λ_H, λ_S, λ_K, λ_F, λ: {0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10};
(2) Scaling parameters S_H, S_S, S_K, S_F: {0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95};

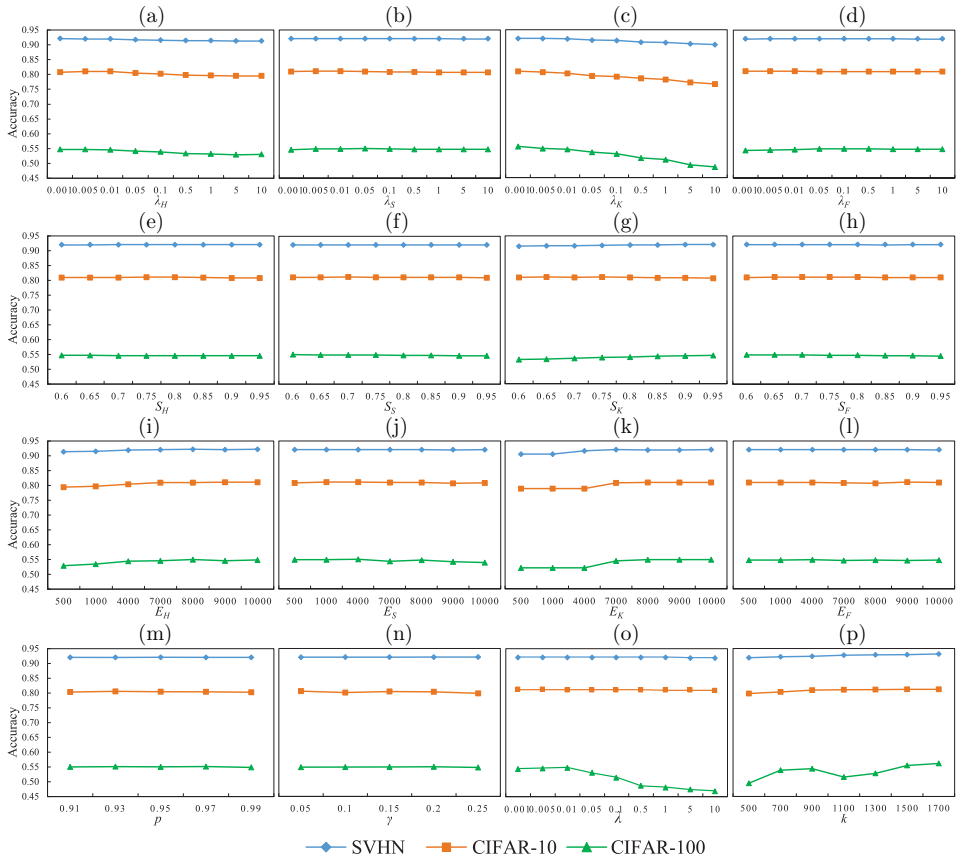


Fig. 9. Parameter sensitivity evaluation for MFBLS on SVHN, CIFAR-10, and CIFAR-100. The x-axis of each subgraph represents the hyperparameter value, and the y-axis represents the test accuracy.

(3) Numbers of enhancement nodes E_H, E_S, E_K, E_F: {500, 1000, 4000, 7000, 8000, 9000, 10000};
(4) Number of cluster centroids k: {500, 700, 900, 1100, 1300, 1500, 1700};
(5) PCA hyperparameter p: {0.91, 0.93, 0.95, 0.97, 0.99};
(6) Hyperparameter γ used to calculate the number of convolution kernels: {0.05, 0.1, 0.15, 0.2, 0.25}.

Figure 9 shows the experimental results on SVHN, CIFAR-10, and CIFAR-100 with different hyperparameter sets. Each subgraph in Fig. 9 describes the influence of the above hyperparameters on the classification performance for the three datasets. The x-axis of each subgraph represents the hyperparameter value, and the y-axis represents the test accuracy. For example, Fig. 9(a) shows the test accuracies for the three datasets using different λ_H values. We can draw the following conclusions from the results shown in Fig. 9:

(1) The blue lines (SVHN) are the flattest lines when the hyperparameters change, followed by the orange lines (CIFAR-10), and then green lines (CIFAR-100). This indicates that MFBLS might more sensitive to changes in hyperparameters for complex datasets.
(2) Accuracies from Figs. 9(a) to 9(n) hardly change when all parameters but λ_H, λ_K, E_H, and E_K change.
(3) Green lines (CIFAR-100) in Figs. 9(o) and 9(p) are most tortuous among all the green lines. It suggests that the parameters k and λ have a relatively significant influence on CIFAR-100.

In summary, Fig. 9 shows that when most of the hyperparameters change, the accuracies for the three datasets vary slightly within a certain range. This indicates that the proposed model is not sensitive to changes in most above hyperparameters.

5. Conclusion

In this paper, we propose an MFBLS model to improve the image classification performance of BLS and its variants while ensuring a relatively short training time. The model has two major characteristics: multi-feature extraction and parallel structure. MFBLS uses a multi-feature extraction method instead of the random mapping in original BLS to significantly improve the image-feature-learning ability. Four carefully selected features, convolutional feature, K-means feature, HOG feature, and color feature, are extracted in MFBLS. In order to facilitate the implementation of the multi-feature extraction method, we propose a parallel architecture for MFBLS. There are four feature blocks and one fusion block in this structure. The feature blocks are parallel to each other, and each uses extracted features directly as the feature nodes in feature block, hence no random feature mapping is needed for feature nodes generation. The fusion block fuses the outputs of the feature blocks by using a proposed stacking with ridge regression strategy to get better results.

Experimental results on SVHN, CIFAR-10, and CIFAR-100 datasets show that the accuracy of image classification of the MFBLS is much better than that of all the BLS and its variants compared, although the training time is slightly increased. What is more, MFBLS outperforms the convolutional DBN both in accuracy and training time, which indicates that our model can even replace some deep networks for learning tasks for complex datasets. Please note that some deep networks like convolutional DBN require pre-training, while there is no need of pre-training for MFBLS. The classification performance of MFBLS is also better than LeNet5, SDT-ELM, and ELM-ARF. We also demonstrate that our proposed parallel structure is more suitable for MFBLS than concatenated structure by experiments and time complexity analysis. In addition, experimental results show that our proposed fusion strategy for the parallel structure outperforms the averaging and voting strategies. Analysis results of parameter sensitivity indicate that MFBLS is not sensitive to changes in most hyperparameters.

In this paper, we have proved that our proposed MFBLS is an effective and efficient model without deep architecture for image classification. However, we suspect that there may be other types of features that can further improve the performance of MFBLS if they are extracted properly for it. In the future, we will focus on how to find/design these features, and more types of features will be tested for different datasets in order to further evaluate the model.

Acknowledgments

This work was supported in part by the Fundamental Research Funds for the Central Universities (2019CDYGZD004); the Chongqing Foundation and Advanced Research Project (cstc2019jcyj-msxmX0622); the Science and Technology Research Program of Chongqing Municipal Education Commission (KJQN201800111), the Sichuan Science and Technology Program (2019YFSY0026), and the Open Fund Project of Chongqing Key Laboratory of Translational Research for Cancer Metastasis and Individualized Treatment.

References

1. V. Bolón-Canedo and B. Remeseiro, Feature selection in image analysis: A survey, *J. Vis. Commun. Image Represent.* **53**(4) (2019) 1–27.
2. C. L. P. Chen and Z. Liu, Broad learning system: An effective and efficient incremental learning system without the need for deep architecture, *IEEE Trans. Neural Netw. Learn. Syst.* **29**(1) (2018) 10–24.
3. C. L. P. Chen, Z. Liu and S. Feng, Universal approximation capability of broad learning system and its structural variations, *IEEE Trans. Neural Netw. Learn. Syst.* **30**(4) (2019) 1191–1204.
4. S. Cheng and G. Zhou, Facial expression recognition method based on improved vgg convolutional neural network, *Int. J. Pattern Recogn. Artif. Intell.* **34**(7) (2020) art. no. 2056003, doi: 10.1142/S0218001420560030.

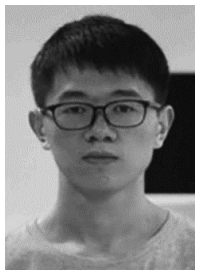
5. A. Coates and A. Y. Ng, Learning feature representations with k-means, in *Neural Networks: Tricks of the Trade*, Lecture Notes in Computer Science, eds. G. Montavon, G. Orr and K.-R. Müller, Vol. 7700. (Springer, Berlin, Heidelberg, 2012), pp. 561–580.
6. E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan and Q. V. Le, Autoaugment: Learning augmentation strategies from data, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (IEEE, Long Beach, CA, USA, 2019), pp. 113–123.
7. N. Dalal and B. Triggs, Histograms of oriented gradients for human detection, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (IEEE, San Diego, CA, USA, 2005), pp. 886–893.
8. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A large-scale hierarchical image database, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (IEEE, Miami, FL, USA, 2009), pp. 248–255.
9. T. Devries and G. W. Taylor, Improved regularization of convolutional neural networks with cutout (2017), https://arxiv.org/abs/1708.04552.
10. I. S. Dhillon and D. S. Modha, Concept decompositions for large sparse text data using clustering, *Mach. Learn.* **42**(1–2) (2001) 143–175.
11. R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification* (John Wiley & Sons, 2012).
12. Y. Garg and K. S. Candan, RACKNet: Robust allocation of convolutional kernels in neural networks for image classification, in *Proc. 2019 Int. Conf. Multimedia Retrieval* (ACM, Ottawa, ON, Canada, 2019), pp. 315–323.
13. I. Goodfellow, D. Wardefarley, M. Mirza, A. Courville and Y. Bengio, Maxout networks, in *Int. Conf. Machine Learning* (IMLS, Atlanta, GA, USA, 2013), pp. 1319–1327.
14. K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (IEEE, Las Vegas, NV, USA, 2016), pp. 770–778.
15. G. E. Hinton, S. Osindero and Y.-W. Teh, A fast learning algorithm for deep belief nets, *Neural Comput.* **18**(7) (2006) 1527–1554.
16. A. E. Hoerl and R. W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems, *Technometrics* **42**(1) (2000) 80–86.
17. B. Igelnik and Y.-H. Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link net, *Neural Netw.* **6**(6) (1995) 1320–1329.
18. J. Jin, Z. Liu and C. P. Chen, Discriminative graph regularized broad learning system for image recognition, *Sci. China Inf. Sci.* **61**(11) (2018) 112209.
19. J. Kittler, M. Hatef, R. P. W. Duin and J. Matas, On combining classifiers, *IEEE Trans. Pattern Anal. Mach. Intell.* **20**(3) (1998) 226–239.
20. A. Krizhevsky and G. Hinton, Convolutional deep belief networks on CIFAR-10 (2010), https://www.cs.toronto.edu/∼kriz/.
21. A. Krizhevsky and G. Hinton, Learning multiple layers of features from tiny images, technical report, University of Toronto, Toronto, 2009.
22. A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (ACM, Lake Tahoe, NV, USA, 2012), pp. 1097–1105.
23. Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, *Proc. IEEE* **86**(11) (1998) 2278–2324.
24. Y. LeCun, F. J. Huang and L. Bottou, Learning methods for generic object recognition with invariance to pose and lighting, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (IEEE, Washington, DC, USA, 2004), pp. 97–104.

25. S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi and J. A. Benediktsson, Deep learning for hyperspectral image classification: An overview, *IEEE Trans. Geosci. Remote Sensing* **57**(9) (2019) 6690–6709.
26. S. Liang, Y. Khoo and H. Yang, Drop-activation: Implicit parameter reduction and harmonic regularization (2020), https://arxiv.org/abs/1811.05850.
27. B. Liu, X. Zeng, F. Tian, S. Zhang and L. Zhao, Domain transfer broad learning system for long-term drift compensation in electronic nose systems, *IEEE Access* **7** (2019) 143947–143959.
28. Z. Liu, J. Zhou and C. P. Chen, Broad learning system: Feature extraction based on k-means clustering algorithm, in *2017 4th Int. Conf. Information, Cybernetics and Computational Social Systems (ICCSS)* (IEEE, Dalian, China, 2017), pp. 683–687.
29. B. S. Manjunath, J.-R. Ohm, V. V. Vasudevan and A. Yamada, Color and texture descriptors, *IEEE Trans. Circuits Syst. Video Technol.* **11**(6) (2001) 703–715.
30. Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, Reading digits in natural images with unsupervised feature learning, in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* (Granada, Spain, 2011), pp. 1–9.
31. Y.-H. Pao, G.-H. Park and D. J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, *Neurocomputing* **6**(2) (1994) 163–180.
32. Y.-H. Pao and Y. Takefuji, Functional-link net computing: Theory, system architecture, and functionalities, *Computer* **25**(5) (1992) 76–79.
33. J. K. Patil and R. Kumar, Color feature extraction of tomato leaf diseases, *Int. J. Eng. Trends Technol.* **2**(2) (2011) 72–74.
34. R. Salakhutdinov and G. Hinton, Deep Boltzmann machines, in *Proceedings of the International Conference on Artificial Intelligence and Statistics* (PMLR, Clearwater Beach, FA, USA, 2009), pp. 448–455.
35. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, in *Int. Conf. Learning Representations* (ICLR, San Diego, CA, USA, 2015), arXiv:1409.1556.
36. K. E. Smith and P. Williams, A shallow learning-reduced data approach for image classification, in *Canadian Conf. Artificial Intelligence* (Springer, Kingston, ON, CAN, 2019), pp. 345–351.
37. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (IEEE, Boston, MA, USA, 2015), pp. 1–9.
38. M. Szummer and R. W. Picard, Indoor-outdoor image classification, in *Proc. 1998 IEEE Int. Workshop on Content-Based Access of Image and Video Database* (IEEE, Bombay, IND, 1998), pp. 42–51.
39. M. Tan and Q. V. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, in *Int. Conf. Machine Learning* (IMLS, Long Beach, CA, USA, 2019), pp. 6105–6114.
40. W. Wang, Y. Yang, X. Wang, W. Wang and J. Li, Development of convolutional neural network and its application in image classification: A survey, *Opt. Eng.* **58**(4) (2019) 040901.
41. Y. Wang and Z. Wang, A survey of recent work on fine-grained image classification techniques, *J. Vis. Commun. Image Represent.* **59** (2019) 210–214.
42. J. S. Wei, J. C. Lv and Z. Yi, A new sparse restricted Boltzmann machine, *Int. J. Pattern Recogn. Artif. Intell.* **33**(10) (2019), art. no. 1951004, doi: 10.1142/S0218001419510042.
43. C. Wu, Y. Li, Z. Zhao and B. Liu, Extreme learning machine with autoencoding receptive fields for image classification, *Neural Comput. Appl.* **32**(12) (2020) 8157–8173.

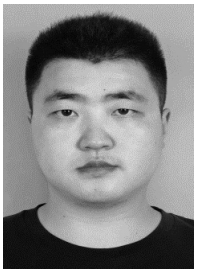
44. M. Xu, J. Cheng, D. W. K. Wong, A. Taruya, A. Tanaka and J. Liu, Automatic atherosclerotic heart disease detection in intracoronary optical coherence tomography images, in *Int. Conf. IEEE Engineering in Medicine and Biology Society* (IEEE, Chicago, IL, USA, 2014), pp. 174–177.
45. L. Xu, A. Krzyzak and C. Y. Suen, Methods of combining multiple classifiers and their applications to handwriting recognition, *IEEE Transactions on Systems, Man, and Cybernetics* **22**(3) (1992) 418–435.
46. F. Yang, A CNN-based broad learning system, in *IEEE Int. Conf. Computer and Communications* (IEEE, Chengdu, China, 2018), pp. 2105–2109.
47. F. W. Yang, H. J. Lin, S.-H. Yen and C.-H. Wang, A study on the convolutional neural algorithm of image style transfer, *Int. J. Pattern Recogn. Artif. Intell.* **33**(5) (2019), art. no. 1954020, doi 10.1142/S021800141954020X.
48. C. Zetzsche, G. Krieger and B. Wegmann, The atoms of vision: Cartesian or polar?, *J. Opt. Soc. Am. A-Opt. Image Sci. Vis.* **16**(7) (1999) 1554–1565.
49. J. Zhao and L. Jiao, Sparse deep tensor extreme learning machine for pattern classification, *IEEE Access* **7** (2019) 119181–119191.
50. Q. Zhou and X. He, Broad learning model based on enhanced features learning, *IEEE Access* **7** (2019) 42536–42550.

Ran Liu received his B.E., M.E., and D.E. degrees in computer science and technology from Chongqing University, Chongqing, China, in 2001, 2004, and 2007, respectively. He worked as a postdoctoral researcher for Homwee Technology Co., Ltd., Chengdu, China, from 2008 to 2010. From 2015 to 2016, he was a Research Fellow with the Schepens Eye Research Institute, Massachusetts Eye and Ear, Department of Ophthalmology, Harvard Medical School, Boston, MA, USA. He is now an Associate Professor at the College of Communication Engineering and College of Computer Science, Chongqing University, China. His research interests include image processing, machine learning, and 3D imaging.

Yaqiong Liu received her B.E. degree in software engineering from Henan University, China, in 2018. She is currently pursuing her M.E. degree from Chongqing University. Her research interests include computer vision and machine learning.

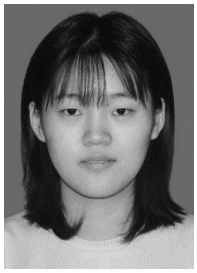


Yang Zhao received his B.E. degree in computer science and technology from Chongqing University, China, in 2018. He is currently pursuing his M.E. degree from Chongqing University. His research interests include computer vision and medical image processing.



Xi Chen received his B.S. degree in information and computing science from Hohai University, China, in 2019. He is currently pursuing his M.S. degree from Chongqing University. His research interests include machine learning and signal processing.

Shanshan Cui received her B.E. degree in computer science and technology from Wuhan Institute of Technology, China, in 2019. She is currently pursuing her M.E. degree from Chongqing University. Her research interests include machine learning and image processing.



Feifei Wang received her B.E. degree in computer science and technology from Zaozhuang University, China, in 2019. She is currently pursuing her M.E. degree from Chongqing University. Her research interests include deep learning and image processing.

Lin Yi received her Bachelor's and Master's degrees in clinical laboratory diagnostics from Chongqing Medical University, Chongqing, China, in 2004 and 2020, respectively. She was a Visiting Scholar with the Schepens Eye Research Institute, Massachusetts Eye and Ear, Department of Ophthalmology, Harvard Medical School, Boston, MA, USA, in 2016. She is currently an associate senior technologist at Chongqing University Cancer Hospital. Her research interests include medical image processing and machine learning.