




CMB-net: a deep convolutional neural network for diagnosis of cerebral microbleeds

Zhihai Lu¹ · Yan Yan² · Shui-Hua Wang² 

Received: 25 June 2020 / Revised: 5 January 2021 / Accepted: 13 January 2021 /

Published online: 5 February 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC part of Springer Nature 2021

Abstract

Cerebral microbleed (CMB) is related to cerebral vascular diseases. In this paper, we propose the use of deep convolutional neural network to implement CMB automatic diagnosis based on brain susceptibility-weighted images (SWIs). First of all, a sliding neighborhood method was employed to get 13,031 samples for training and testing. Then, an 18-layer CMB-Net was designed to classify the samples as CMB or non-CMB. The CMB-Net was trained by RMSprop based on the five-fold cross-validation. The total running time of the five-fold cross-validation was merely 184.79 s, and the average testing accuracy reached 98.39%, which was better than several recently published methods. The results suggested that our CMB-Net was accurate in detecting CMB.

Keywords Cerebral microbleed · Susceptibility-weighted image · Computer aided diagnosis · Convolutional neural network

1 Introduction

Cerebral microbleed (CMB) is usually caused by the leaking of small vessels in brain, which is believed to be related to cerebral vascular diseases. Currently, the diagnosis of CMB is dependent on medical images. Susceptibility-weighted imaging (SWI) combines amplitude and phase information to enhance the contrast of CMB to improve the sensitivity of feature detection. Therefore, SWI yields images with better contrast than magnetic resonance imaging (MRI),

✉ Shui-Hua Wang
shuihuawang@ieee.org

Zhihai Lu
luzhihai@nynu.edu.cn

Yan Yan
yanyan899@outlook.com

¹ School of Education Science, Nanjing Normal University, Nanjing, Jiangsu 210023, China

² School of Informatics, University of Leicester, Leicester LE1 7RH, UK

which is beneficial for CMB detection, as CMBs appear like small round dots. However, it's difficult and time consuming to label the CMBs manually, and manual analysis suffers from high inter-observer and intra-observer variance. Hence, developing computer aided diagnosis (CAD) systems for CMB is of significance, which can serve as reference and verification to help doctors. So far, CAD systems for CMB detection are based on computer vision techniques and machine learning algorithms to implement automated image classification.

Barnes, et al. [1] suggested to use a statistical thresholding method to get the CMB candidates from brain SWI. Then, a support vector machine (SVM) was trained to identify the true CMBs from false CMBs. The sensitivity obtained from their experiment was 81.7%. Kuijf, et al. [15] proposed to employ radial symmetry transform (RST) for automated CMB classification. They performed the RST on two echoes of MRI sequence to identify the CMBs. Then, the results were checked manually. Their sensitivity was 71.2%, but the diagnosis time of their system was less than human rating. Bian, et al. [2] introduced 2D fast RST to generate potential CMBs. The false positives were removed by the geometric features. Their method achieved a sensitivity of 86.5%. Fazlollahi, et al. [4] used a Laplacian of Gaussian algorithm to generate sphere objects as CMB candidates. Then, a set of shape features were extracted from these candidates and sent into a cascade of random forests for classification. The best sensitivity of their method was 93%. Hou and Chen [11] designed a sparse autoencoder to generate features from SWIs and trained a four-layer deep neural network for classification, which produced a sensitivity of 93.20%. Hou [10] compared the classification performance using Logistic sigmoid function, rectified linear unit, and leaky rectified linear unit as activation functions for single hidden layer feedforward neural network. The results on cross-validation revealed that the sensitivity was 93.05%. Zhang, et al. [30] constructed a seven layer deep neural network and achieved sensitivity of 95.13% for CMB classification. Chen, et al. [3] proposed to use 3D residual neural network to classify CMB from non-CMB. Hong [9] proposed a fully optimized convolutional neural network (CMB), and achieved an accuracy of 98.32%. Later, Hong [8] leveraged ResNet and transfer learning to detect CMB and achieved accuracy of 97.46%.

The classification performance of the above methods was good, but there are also problems. For example, the RST is a frequently used tool, but it belongs to a type of handcrafted features. Hence, in this paper, we proposed to leverage deep CNN to classify CMB and non-CMB. In our CMB-Net, the image representations can be obtained automatically, and the structure of the CMB-Net is also smaller than famous deep CNNs like ResNet, etc. The proposed CMB-Net achieved better classification performance than several existing methods. Meanwhile, the time for once training was only 36.96 s, which was fast. The rest of this paper is as follows. Section 2 provides the CMB dataset in our experiment and gives the detailed formulation of our CMB-Net, then, Section 3 is about the performance measurements for evaluation. The results and discussion are presented in Section 4 and the conclusion is in Section 5.

2 Methods

Deep convolutional neural networks have achieved exciting improvements in image recognition, and a plenty of famous architectures have been proposed, such as AlexNet [14], VGG [23], ResNet [6], MobileNet [12], etc.. Deep CNN has become the most cutting-edge technique in computer vision, which has been widely applied in practical situations. For two reasons, we didn't use those famous CNN models. One is that those models were pre-trained

on ImageNet dataset which contains 1000 types of samples, but our CMB detection is a binary classification problem which doesn't require a CNN model of that deep size, and deeper structure has more parameters to train. More importantly, those CNN models are trained by images of 227×227 , but our samples are in size of 41×41 . This difference in size poses a big challenge to leverage transfer learning for CMB detection. Resizing images to that big size may introduce errors. Therefore, we want to construct a new CNN model for CMB classification. Our proposed model is called CMB-Net.

Conventional CNNs usually include convolution layer, pooling layer, and fully connected layer, and we also added batch normalization and dropout in our CMB-Net to get better convergence speed and higher generalization ability. The formulation of these techniques is presented in this section. Our proposed CMB-Net can be transferred to tasks in computer vision fields.

2.1 Material

We obtained 20 SWI samples in total with 10 for cerebral autosomal dominant arteriopathy with subcortical infarcts and Leukoencephalopathy (CADASIL) and 10 for normal control. Since CMB are distributed in the SWIs, there can be multiple CMB marks in one image slice, so we proposed to use sliding neighborhood method to generate CMB and non-CMB samples from the original SWIs. We used a window to slide on the SWI to get sub-images, as shown in Fig. 1. The labels of the obtained samples are determined by their central pixels. If the central pixel is in a CMB region, that sample is labeled as CMB, and if the central pixel is not in CMB, the sample is labeled as non-CMB. By this sliding neighborhood algorithm, we obtained 13,031 samples in size of 41×41 , with 6407 CMBs and the rest 6624 non-CMBs.

2.2 Basic mechanisms of CNN

The meaning of CNN is to extract features of images with certain models, and then classify, identify, predict or make decisions on the images according to the features. In this process, the most important step of feature extraction is to conduct iterative training on CNN to extract features that can distinguish images to the greatest extent, so that the model can accurately recognize the image, and thus realize the large model of high-quality diagnosis system.

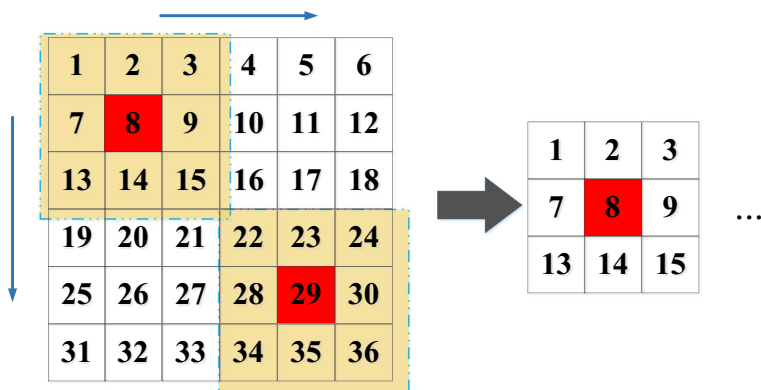


Fig. 1 Sample generation by sliding window

Traditional computer vision methods often use handcrafted image features, but CNN provides a new feature learning mechanism. The convolution operations in CNN are capable of extracting image features automatically. The image representations can be generated from low level to high level gradually in CNN. These automated feature learning are better than handcrafted features for classification. Because manual feature extraction relies on the experience of designer, automatic feature extraction relies on huge amounts of data. Meanwhile, the local perspective also enables parameter sharing, which reduces the total parameter number in deep CNNs. The convolution operation can be expressed as

$$\text{convolution}(M, K) = \sum_I \sum_J M(i-m, j-n) K(m, n) \quad (1)$$

where M in size of (I, J) denotes input image, and K in size of (m, n) stands for the kernel. The kernel scans the image line by line to generate a new feature map. The stride is step length that the kernel moves every time, which is pre-defined. Zero padding is often used in convolution with the aim to get information from the borders and corners.

2.2.1 The pooling layer used in CMB-NET

Pooling layers in a CNN are used for feature reduction. Pooling operation employs a sliding window as the local perspective to preserve the most important information from the feature maps and substantially reduce the feature volume. For example, the max pooling method simply preserves the maximum value in the local perspective to form the reduced feature maps [5, 18, 25, 31].

A pooling layer is used to sample the input image and can be understood as a filter. The usual form is to sample the feature map in individual slices along each depth with a stride of 2 and a size of 2×2 and a constant depth. The process is illustrated as follows:

- i. The volume of an input with width, height, and depth is obtained, with a size of $W_1 \times H_1 \times D_1$.
- ii. Set two hyperparameters: the spatial range R and the stride L .
- iii. Extract a volume of size $W_2 \times H_2 \times D_2$, where

$$W_2 = (W_1 - R) / L + 1 \quad (2)$$

$$H_2 = (H_1 - R) / L + 1 \quad (3)$$

$$D_2 = D_1 \quad (4)$$

- iv. Zero parameters are used to fill the pooling layer.

Dimensionality reduction sampling as a feature of pooling layer in neural network can reduce the parameter range and retain the feature invariance as much as possible. The faster speed of the neural networks is due to this dimension reduction property. Average pooling and max pooling can be used to obtain the average value and maximum value of the neighborhood of

feature points respectively. The average pooling mainly considers the overall feature of information and can reduce estimates variance owing to the limited neighborhood size, allows as much reserved information to be passed on to the next level as possible. Max pooling can reduce the influence of convolutional layer parameters on the deviation of the estimated mean, and it can screen features to enhance the prominence of features and reduce noise.

Therefore, average pooling is used before classification in general because of its global consideration characteristics [16]. Compared with the average pooling, max pooling applied to the middle layer of the model by eliminates redundant information and considers local features on average.

The fully connected layers are the most common structure in artificial neural networks [13, 19, 20, 26]. Every node in the layer is connected to all the nodes in the adjacent layer. Fully connected layers are usually in the last layers in a CNN, they serve as the classifier, which receives the features from early layers and output the labels.

The interval covariate shift is a problem in the training of CNNs. The distributions of the output in different layers are different, and when the model gets deeper, training will be difficult. The convergence of training will be dependent on the good initialization of the parameters. Therefore, batch normalization was invented to handle this problem. It normalizes the output of its early layer so that the normalized data can be kept within the activation region and the training can be fast.

2.2.2 Dropout training mechanism for CMB-net

Dropout is a mechanism in training of fully connected layers. During the training process, part of neurons of the fully connected neural network in Fig. 2 are randomly hidden, as shown in Fig. 3.

For a machine learning model, the trained model with a mass of parameters but few training samples is prone to overfitting. Dropout, as an effective tool, can reduce the possibility of abandoning long-trained model which caused by model overfitting to a certain extent. Dropout randomly disregards some connections in training so that the remaining connections can be trained efficiently. The generalization performance of the entire CNN can be improved. In

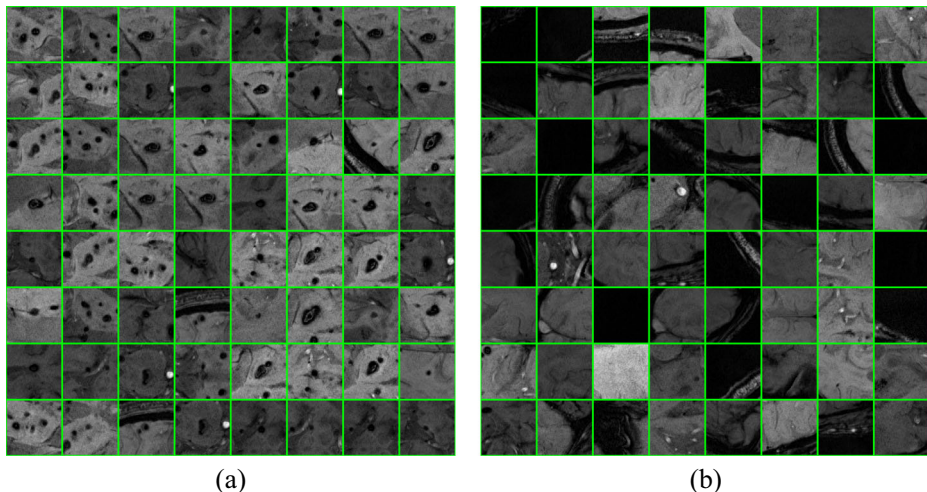
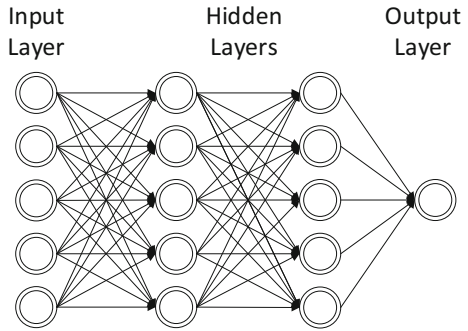


Fig. 2 **a** CMB and **b** non-CMB samples

Fig. 3 Standard neural network

other words, in the forward propagation, half of the hidden layer nodes in each training batch will stop working with a certain probability to reduce the interaction between hidden layer nodes, make the model has strong generalization ability, and prevent the model from precluding the possibility of correct detection because of individual specific circumstances.

The dropout process begins with the random deletion of half of the hidden neurons, leaving the input and output layers intact. In the second step, the neural network with hidden nodes removed is used for forward propagation of input parameters, and then the result of loss is obtained by backpropagation of neural network. The weights and biases of small batch samples are obtained through the process of stochastic gradient descent. Finally, the neural network restores the deleted neurons, temporarily deleting a subset of the random half of the hidden layer, and forward and backpropagation of small batch samples. This process means that every time the sample used for the input network updates its weights and biases, the extraction of the hidden nodes during the training is random. The entire model dropout process can be viewed as the average result of the model. It has the characteristics of unsupervised pre-training. Each input sample corresponds to a different network structure, which is equivalent to combining and averaging several different neural networks. Meanwhile, share the weights and biases gained during the training.

The randomness of Dropout reduces the probability of two neurons appearing at the same time in each training, which can improve the neural network learning process to avoid making one-sided predictions owing to over-reliance on a fixed structure of the network, so as to prevent excessive decisiveness and inefficient judgment.

Take a neural network with T hidden layers for example [24]. t belongs to the set $\{1, \dots, T\}$ is a random number of layers. Set the input vector and output vector of t layer as $u^{(t)}$ and $v^{(t)}$ respectively. The weights and biases of the t layer are expressed as $w^{(t)}$ and $b^{(t)}$. Define S to be any activation function. In a standard feed-forward neural network without dropout, the t layer set is $\{1, \dots, T-1\}$, and the random hidden unit is r . The formula for this standard feed-forward neural network is shown as below:

$$u_r^{(t+1)} = w_r^{(t+1)} v^t + b_r^{(t+1)} \quad (5)$$

$$v_r^{(t+1)} = S(u_r^{(t+1)}) \quad (6)$$

A dropout model can be represented by the following formula. Where the probability p vector is generated by Bernoulli, that is, for any t layer randomly generated a 0, 1 vector, each of which has a probability of $o = 1$.

$$p^{(t)} \sim \text{Bernoulli}(o) \quad (7)$$

$$\tilde{v}^{(t)} = p^{(t)} * v^{(t)} \quad (8)$$

$$u_r^{(t+1)} = w_r^{(t+1)} \tilde{v}^t + b_r^{(t+1)} \quad (9)$$

$$v_r^{(t+1)} = S(u_r^{(t+1)}) \quad (10)$$

The vector p is sampled and multiplied by the output $v^{(t)}$ of that layer to produce a simplified output $\tilde{v}^{(t)}$ that is used as the input of the next layer. Suppose the number of network neurons in a certain layer is 500, and the output value of the activation function is $v^{(1)}, \dots, v^{(500)}$, then the value of about 400 of the 500 neurons in this layer will be set to 0 after the assumption of 0.8 ratio of dropout. This process applies to each layer. When testing the model, the weight parameter of each neurons multiplied by probability o is narrowed to the formula:

$$w_{test}^{(t)} = o \cdot w^{(t)} \quad (11)$$

Currently, dropout can be used to prevent overfitting on large networks with fewer data sets, whereas using dropout on small networks or with more data sets may increase training time.

2.2.3 Activation function ReLU for CMB-net

In order to overcome the problem caused by gradient disappearance, ReLU is used to replace the two commonly used activation functions Sigmoid and Tanh.

The goal of the activation function is to preserve and map the features of the activated neuron through the function. In classical neural networks, sigmoid function is often employed as the activation function to provide non-linearity. The popularity of sigmoid activation function owing to it compresses the input into a range of 0 to 1, which corresponds to the range of probabilities. The formula of sigmoid function is:

$$\text{sig}(x) = \frac{1}{1 + e^{-x}} \quad (12)$$

In fact, the tanh function is an improvement on the sigmoid function. The tanh function is entered into the interval $(-1, 1)$ with a center of 0. Its activation value is to some extent a normally distributed input that is already at the next layer. The formula of tanh function is:

$$\begin{aligned} \text{Tanh}(x) &= \frac{2}{1 + e^{-2x}} - 1 \\ &= 2\text{sig}(2x) - 1 \end{aligned} \quad (13)$$

Unfortunately, although the sigmoid and tanh functions of nonlinear activation functions are widely used [7], they all share a common supersaturation problem with the deepening of the structure. The gradient vanishing poses a big challenge for the training, which means the

activations are likely to be located in the saturation regions of sigmoid and tanh function. The peaks of the two functions limit their global sensitivity, and only sensitive to local changes. However, in large networks, these nonlinear activation functions will affect the transfer of useful gradient information by the neural network itself. The gradient disappearance means that the error propagation decreases with the increase of layers, hence the machine learning algorithm must continue to adjust the weight to improve the model performance, and the complex adjustment reduces the training efficiency of the network.

The ReLU activation function solves the gradient disappearance problem of sigmoid and tanh activation functions. Therefore, in deep CNN, rectified linear unit (ReLU) is more preferable as activation functions. The ReLU can be expressed as:

$$\begin{aligned}\text{ReLU}(x) &= \max(0, x) \\ &= \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}\end{aligned}\quad (14)$$

The principle of ReLU activation function is to preserve the positive and convert all negative values to zero. It tries to explore how to use a matrix with a majority of 0 to try to express the data characteristics of 1. As a consequence, the problem of gradient disappearance is solved. Because of the sparsity of the function, there is no complex mathematical operations, which leads to the simplification of the model. The reduction of noise and the better prediction ability, and the faster convergence of the neural network, which means that the training and running time of the model is shortened. Nevertheless, it may lead to the “one-sided” result that any unit with negative input neuron activation value of 0 may not activate. Comparing to sigmoid and tanh activate functions, this method is faster and better.

2.2.4 Softmax classifier used for CMB-net

In the selection of classifiers in the image classification, compared with the commonly used support vector machine classifier, Softmax classifier has the advantage of calculating the possibility of all classification tags. Softmax classifier can be understood as a generalization of logistic regression classifier for multiple classification problems [17, 27–29]. Compared to linear regression, its output unit changes from one to multiple.

The calculation principle of Softmax is an evaluation of normalized data. Softmax classifier evaluates the output vector as an unnormalized logarithmic probability and the data is normalized by division so that the sum of these probabilities is 1.

Suppose an input image has two probability values of α and β in two classifications, and $\alpha > \beta$. If the maximum value is defined as the result, the image will be classified directly into α , which is the limitation of classification. In other words, for the sake of measure the error between the discrete value and the output value in the uncertain range, softmax can be used to calculate the probability of both α and β and transform the output value into the probability distribution with positive values, and a sum of all probabilities of 1. This is an assessment of the relative probability of the result.

Softmax classifier is usually bundled with cross-entropy. As it was essentially used to measure the similarity of the two probability distributions, cross-entropy was used in combination with softmax to speed up calculations. The process can be interpreted as Softmax standardizing the classification output into a probability distribution, while cross-entropy compares the similarity between the predicted classification and the true result.

Specifically, in the output layer, softmax function is used to map its input into probabilities:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (15)$$

where x_i denotes the i^{th} element and n is the dimension of the vector.

We assume that the input x_i is a series of eigenvectors, and that the probability of a certain class g in the set of predicted total classes G is P , and that the sum of all predicted classes is 1.

$$P(y = g|x_i) = \frac{e^{x_i * w_j}}{\sum_{g=1}^G e^{x_i * w_g}} \quad (16)$$

where, $y = g$ represents the restricted evaluation category, and w_j and w_g represents the weight of x_i and classes G respectively. The probability of the x_i value obtained in g classification is higher than that of other classifications. The more features belonging to g set, the greater the probability of the output classification belonging to g . To the extent that the components of this map approach 1, and the rest approach 0.

In general, Softmax classifier keeps updating the loss value during the calculation. The correct classification of images has a higher probability, while the wrong classification has a lower probability, and the loss value tends to be smaller.

2.3 Proposed CMB-net

We designed the 18-layer CMB-Net, and the detailed information is listed in Table 2. There are three convolutional layers and two max pooling layers in CMB-Net to learning feature representation from the images.

2.3.1 Use batch normalization in proposed CMB-net

Batch Normalization can be understood as the normalization of data, which is frequently used in the middle layer of the neural network. It preprocesses the data by shifting and scaling the data. After batch processing, the gradient descent algorithm is easier to select the appropriate learning rate, and the descent process will be more stable.

In deep learning, deeper neural networks are often used to train complex problems. Due to the high correlation and coupling between the middle layers, it is difficult and complex to train the deep neural networks. In the process of adjusting the parameters, we need to try different methods to accelerate the convergence of the model.

The parameters in the neural network are constantly updated with the gradient descent as the training progresses. A phenomenon called Internal Covariate Shift is used to explain why model training becomes difficult. In backpropagation, the weight update of each layer is adjusted by loss function under the condition that other weights are not changed. In other words, the ownership weight is updated in the meantime in each backpropagation process, and the change of parameters leads to the change of the input distribution of each layer, hence affecting the upper network to adapt to these changes, making model training difficult.

Batch Normalization can be used to control the input of each layer and solve the problem of unstable data of each layer. At every stochastic Gradient Descent, the Batch Normalization layer tries to normalize the corresponding activation.

Firstly, a linear transformation is carried out for the inputs of each layer. The distribution of the input values of any neuron in the neural network of each layer is processed according to the standard normal distribution with a mean value of 0 and a variance of 1. The scale and shift operation are then performed on the transformed input value. Each neuron added the scale and shift parameters acquired through training and learning, and the activation value of the input value after Batch Normalization transformation is inversely transformed. This process tries to find a good balance between linearity and nonlinearity through the scale and shift of standard normal distribution.

The Batch Normalization layer is generally added between each full connection layer and the activation function. In order to transfer data values effectively, the distribution of calculated values is vital for the activation function. Batch Normalization realigns the data distribution. We assume that the input value of a Batch for a layer of neural network is $D = \{d_1, \dots, d_c\}$, Where d_i is a sample and c is batch size. The formula begins by finding the mean value of the elements in a small batch (μ_{miniB}):

$$\mu_{miniB} = \frac{1}{c} \sum_{i=1}^c d_i \quad (17)$$

The calculation formula of batch standard deviation represented by σ_{miniB}^2 is as follows:

$$\sigma_{miniB}^2 = \frac{1}{c} \sum_{i=1}^c (d_i - \mu_{miniB})^2 \quad (18)$$

The output of the previous activation layer is then normalized by subtracting the average value of batch and dividing by the batch standard deviation.

$$d_i^\wedge = \frac{d_i - \mu_{miniB}}{(\sigma_{miniB}^2 + \varepsilon)^{\frac{1}{2}}} \quad (19)$$

The normalized value is denoted by d_i^\wedge , and ε is a small difference value.

To compensate for the nonlinear performance of the network owing to the loss of offset, the normalized elements are scaled and offset to realize the identity transformation to the original distribution.

$$z_i = \Upsilon_i \cdot d_i^\wedge + \theta_i \quad (20)$$

where z_i represents the final output value. If Υ_i and σ_{miniB}^2 are equal, and θ_i is equal to μ_{miniB} then the identity transformation is realized.

Pseudo-code for the entire process of batch standardization is shown in Table 1.

In this study, three batch normalization layers are employed to normalize the activations. Although the normalization relies on the size of the batch, it has been improved by adding an affine transform to its original operations. In addition, Batch Normalization reduces the complexity of tuning parameters. Furthermore, it can speed up the model training, use a higher learning rate, and increase the generalization ability to some extent.

2.3.2 Structure of proposed CMB-net

There are ten layers with trainable parameters. The schematic diagram of CBB-NET is shown in Fig. 4, and the parameter information of each layer is shown in Table 2.

The kernel size of the convolutional layers is set as 3×3 , because our input image size is only 41×41 , and we want to generate texture features by smaller kernel size. In fact, a small

Table 1 Pseudo-code for batch normalization

The Process of Batch Normalization

Input: Values of d over a mini batch: $D = \{d_1, \dots, d_c\}$

Parameters to be learned: γ and θ

Output: $\{z_i = \text{BN}_{\gamma, \theta}(d_i)\}$

The normalization process:

$$\mu_{\text{miniB}} \leftarrow \frac{1}{c} \sum_{i=1}^c d_i \quad // \text{mini batch mean}$$

$$\sigma_{\text{miniB}}^2 \leftarrow \frac{1}{c} \sum_{i=1}^c (d_i - \mu_{\text{miniB}})^2 \quad // \text{mini batch variance}$$

$$d_i^\wedge \leftarrow \frac{d_i - \mu_{\text{miniB}}}{(\sigma_{\text{miniB}}^2 + \varepsilon)^{\frac{1}{2}}} \quad // \text{normalize}$$

The inverse of the normalization process:

$$z_i \leftarrow \gamma_i \cdot d_i^\wedge + \theta_i \equiv \text{BN}_{\gamma, \theta}(d_i) \quad // \text{scale and shift}$$

convolution kernel can greatly reduce the number of parameters and computational complexity compared with a large convolution kernel with a large receptive field. Convolution kernel filtering is equivalent to a linear calculation. Sampling and pooling after convolution kernel are both for abstracting local features and enhance the ability of the neural network to learn features. Large convolution kernel is easy to cause the loss of deep network information, and a small convolution kernel makes it difficult to reduce the size, resulting in large computation. Through the combination of convolution and pooling, more advanced features can be extracted in depth.

There are three Batch Normalization (BN) layers, all of which are placed behind the convolutional layer and in front of the nonlinear activation function. The BN layer is mainly used to transfer the normalization of the input of the hidden layer, and to alleviate the gradient attenuation problem existing in the nonlinear function in the meantime. As the activation

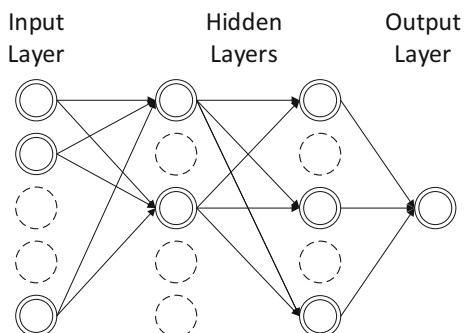
Fig. 4 After applying dropout

Table 2 Information of proposed CMB-net

Name	Type	Activations	Trainable parameters
Input	Image input	41×41×1	—
Conv1	32 3×3×1 convolution, stride 1×1, padding 2×2	43×43×32	Weights 3×3×32 Biases 1×1×32
BN1	Batch normalization	43×43×32	Offset 1×1×32 Scale 1×1×32
Relu1	ReLU activation	43×43×32	—
Maxpool1	3×3 max pooling, stride 1×1, padding 0×0	41×41×32	—
Conv2	128 3×3×64 convolution, stride 1×1, padding 2×2	43×43×64	Weights 3×3×32×64 Biases 1×1×64
BN2	Batch normalization	43×43×64	Offset 1×1×64 Scale 1×1×64
Relu2	ReLU activation	43×43×64	—
Conv3	convolution, stride 1×1, padding 2×2	43×43×64	Weights 5×5×64×64 Biases 1×1×64
BN3	Batch normalization	43×43×64	Offset 1×1×64 Scale 1×1×64
Relu3	ReLU activation	43×43×64	—
Maxpool2	7×7 max pooling, stride 1×1, padding 0×0	37×37×64	—
Fc1	64 fully connected nodes	1×1×64	Weights 64×87,616 Biases 64×1
Relu3	ReLU activation	1×1×64	—
Dropout	50% dropout	1×1×64	—
Fc2	2 fully connected nodes	1×1×2	Weights 2×64 Biases 2×1
Softmax	Softmax activation	1×1×2	—
Output	Output label	—	—

function, ReLU is placed in front of BN, the BN layer is unstable, which affects the performance of the model. By placing BN in front of the activation function, the problem of complete suppression of the activation value of a random layer can be prevented, and the gradient can be prevented from disappearing. Therefore, BN is appropriate to be placed in front of the activation layer after the convolutional layer to accelerate convergence.

Maxpooling is used as the sampling layer to filter and select features of the input information to improve the spatial invariability to a certain extent. Maxpooling is to reduce the size of images, increase the convolution kernel receptive field, and simplify the amount of feature Map data. In this neural network, the window size of ‘Maxpool2’ is 7×7 in order to reduce the number of features. It contributes to prevent overfitting and reduce a large amount of computation caused by multiple parameters.

The fully connected layer is the classifier in the neural network, which integrates the local feature map of learning to obtain the global feature information and conduct classification. The dropout layer is added between the two full connection layers to prevent overfitting and improve the model generalization ability. Because of the redundant parameters of the full connection layer, Softmax was chosen to receive the sample global feature information to calculate the probability of the category of the input image.

2.3.3 Use RMSProp to train proposed CMB-net

The most popular optimization algorithm is gradient descent. The optimizer is mainly used for stable and high-speed convergence of network loss functions. With the continuous

improvement of the gradient descent algorithm, the Root Mean Square Prop (RMSProp) algorithm becomes one of the stable algorithms [22].

As an earlier algorithm, Stochastic gradient descent (SGD) can disassemble data into small batches for training, but it needs a suitable learning rate and is easy to converge to the local optimal. With added momentum method, SGD alleviates the problem of the oscillation near the local extreme value. Although the minimum value can be obtained, it takes a long time to converge the model. The proposed of Adaptive Gradient Algorithm (Adagrad) improves the robustness of SGD because it has the function of adaptive adjusting the learning rate. However, with the beginning of training, the accumulation of square gradient leads to the decline of learning rate, which leads slowly convergence in the later period and even causes the end of training ahead of schedule. It also relies on a manually set global learning rate. Another disadvantage of Adagrad is that it relies heavily on the global learning rate set manually.

RMSProp is a proposed adaptive learning rate method to solve the steep decline of Learning rate in Adagrad [21]. When running the calculation, Adagrad accumulates all the squares of the gradients before, while RMSProp only calculates the corresponding average value. Therefore, the RMSProp method can alleviate the problem that the learning rate of Adagrad algorithm decreases rapidly.

The RMSProp method mainly uses the square weighted average to calculate the cumulative gradient, and the early gradient information is discarded in the calculation process, which means that the farther the gradient is from the current, the smaller the weight of learning rate reduction.

An objective function $F(\varphi)$, which is expected to be minimized. In the way of gradient descent, the objective function $\nabla_{\varphi} F(\varphi)$ is updated in the opposite direction of parameter φ , and the updated parameters are represented as $\Delta\varphi_a$. The formula of RMSProp gradient calculation is as follows:

$$\varphi_{a+1} = \varphi_a - \Delta\varphi_a \quad (21)$$

Where a is the time. $\Delta\varphi_a$ is represented as the updated attenuation parameter,

$$\Delta\varphi_a = -\frac{\lambda}{(q_a + \epsilon)^{\frac{1}{2}}} \odot h_a \quad (22)$$

ϵ is a small constant and avoids the denominator being divided-by-0. h_a here is the gradient of φ at time a .

$$h_a = \nabla_{\varphi} F(\varphi) \quad (23)$$

The square gradient at time a is shown as q_a (Initialize the accumulator variable $q_a = 0$), and the cumulative square gradient is shown as q_{a+1} ,

$$q_{a+1} = cq_a + (1-c)h_a \odot h_a \quad (24)$$

The RMSProp solves the problem of Adagrad training ending early in its later stages, although it relies on the overall learning rate. In fact, in the application of sparse data, adaptive gradient descent shows its excellent optimization ability. For deep and complex neural networks with demand convergence speed or intention training, adaptive algorithms can bring better results than stochastic gradient descent.

3 Performance evaluation

3.1 Cross-validation

The evaluation of our CMB-Net was based on five-fold cross-validation (CV). In five-fold CV, the entire dataset is divided into five groups of data in similar size. Then, every time, a group of data is selected for testing and the rest four groups serve as training set. The training and testing is run for five times so that the whole dataset was used for testing once. The diagram of five-fold CV is given in Fig. 5.

3.2 Indexes

We used the following five indexes to evaluate the classification performance of our model.

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad (25)$$

$$\text{Specificity} = \frac{TN}{TN + FP} \quad (26)$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (27)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (28)$$

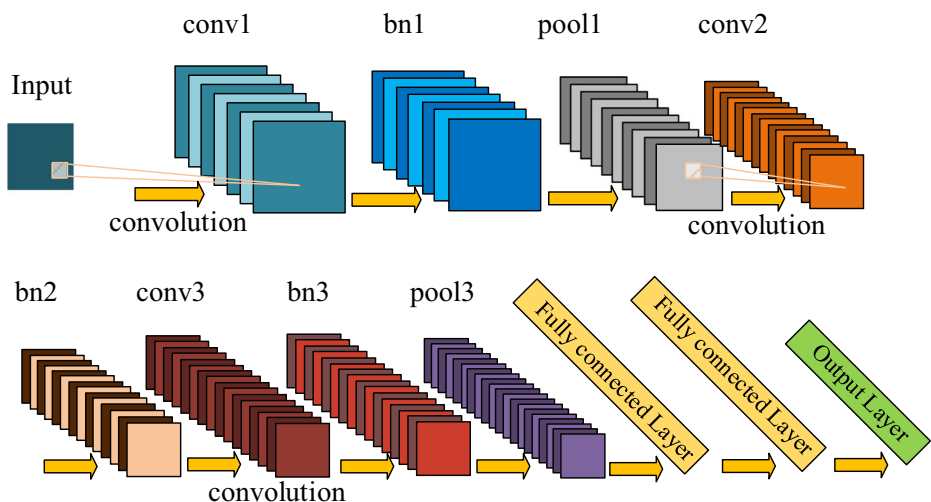


Fig. 5 Schematic diagram of proposed CMB-net

Table 3 Parameter settings

Name	Value
Batch size	200
Max epochs	2
Learning rate	1e-4
L ₂ regularization factor	1e-3

$$F1 \text{ score} = 2 \times \frac{\text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}} \quad (29)$$

In which the *TP*, *TN*, *FP*, and *FN* are the abbreviated forms of true positive, true negative, false positive, and false negative, respectively.

4 Experiment results and discussion

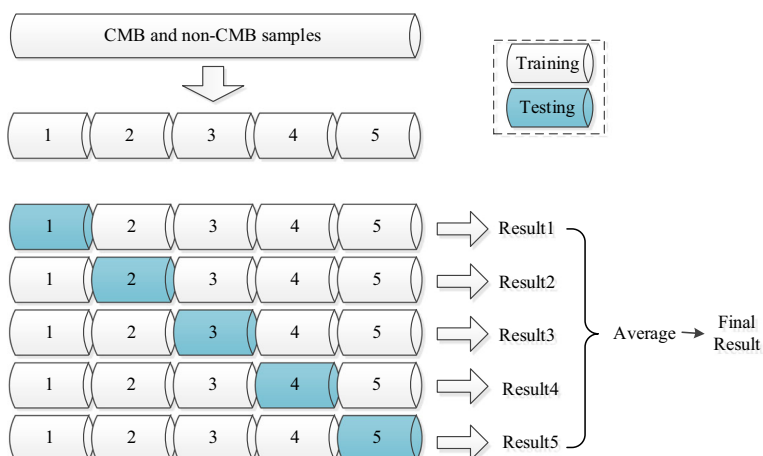
4.1 Experiment settings

Our CMB-Net was implemented on MATLAB R2019b with deep learning toolbox. The training and testing were carried out on a personal computer with CPU i7 7700HQ, RAM 16GB and GPU GTX1060. The hyper-parameter settings are listed in Table 3.

We set the max epochs as only 2 to prevent overfitting, and the CMB-Net can converge within 2 epochs. The batch size is 200 because our training set contains over 10,000 samples. The initial learning rate and regularization factor were default setting.

4.2 CMB-net training and testing

Figure 6 gives the training process of CMB-Net. It can be seen that the initial value of accuracy is about 50% and the loss value is about 1. At the beginning of the training process, the training accuracy and verification accuracy are rapidly improved, while the training loss and verification loss

**Fig. 6** Five-fold cross-validation

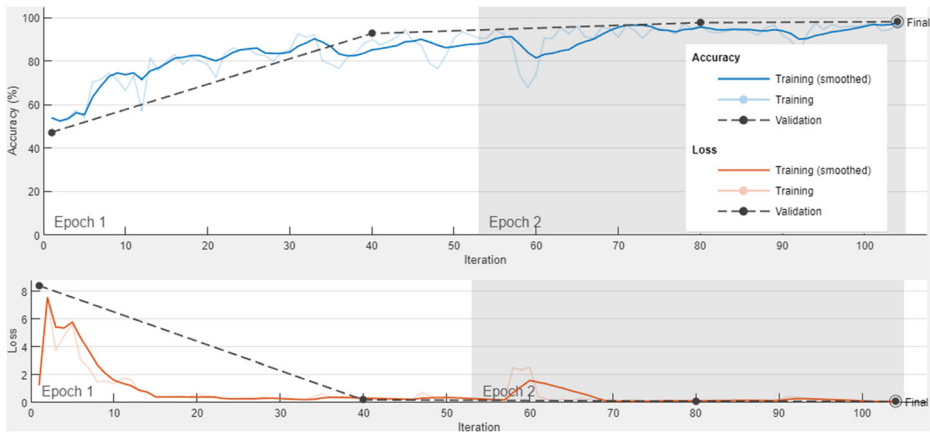


Fig. 7 Once training of CMB-Net

are sharply reduced. Iterations from 0 to 15 show the accuracy curve oscillating wildly, with losses peaking at 8. With the beginning of 20 iterations, the accuracy curve increased to 95% and the loss curve decreased to about 0.2. However, in the iteration from 47 to 60, the loss increased slightly, the accuracy decreased, and the fitting line deviated. The accuracy curve and the loss curve began to stabilize at iteration 60. Finally, by the end of 100 iterations, the best accuracy is approximate 100% and the loss is close to zero.

In order to minimize the loss function and obtain optimized parameters, it was not enough to optimize the learning process with a single epoch. As the number of epochs increased, the updating times of weights in the neural network also increased, and the curve of the neural network changed from underfitting to overfitting. As shown in Fig. 6, it can be inferred that our CMB-Net converged fast within 2 epochs. The training and testing accuracies were both near 100%. Therefore, the number of two epochs met the training requirements.

4.3 Visualization of CMB-net

We also present the convolutional kernels of the first convolutional layer in CMB-Net, which were learnt from the CMB dataset. The plot is given in Fig. 7. There are 32 kernels in size of 3×3 . It can be found that the kernels have captured typical patterns of CMB and non-CMB, so that the CMB-Net could achieve good classification performance.

Table 4 Classification performance of CMB-Net based on five-fold CV

Run	Sensitivity	Specificity	Accuracy	Precision	F1 score
1	98.68%	98.86%	98.77%	98.84%	98.76%
2	97.69%	98.47%	98.08%	98.45%	98.07%
3	97.91%	98.17%	98.04%	97.99%	97.95%
4	98.52%	98.04%	98.27%	97.98%	98.25%
5	98.10%	99.46%	98.77%	99.46%	98.78%
Average	98.18%	98.60%	98.39%	98.54%	98.36%

Table 5 Running time

Model	Time of five-fold CV	Time for once training and testing
CMB-Net	184.79 s	36.96 s

4.4 Classification performance

The final classification performance of our CMB-Net is provided in Table 4. From the results of five operations based on the evaluation of five indicators, it can be found that the fluctuation of the data is about 97% to 98%, and there is basically no particularly large fluctuation, which proves that our model tends to be stable in classification detection to some extent. From the average performance, we can find that our CMB-Net yielded over 98% for all these five indexes, which means that our model is accurate and robust in classification of CMB and non-CMB.

4.5 Time analysis

Deep CNN models can achieve good classification accuracy but the training time is usually too long. However, our CMB classification is a binary classification problem, so we proposed to use a relatively shallow CMB-Net to solve this problem. Our CMB-Net is simple but effective, and the training time is also short. The average training and testing time for once was only 36.96 s, as shown in Table 5.

4.6 Comparison with state-of-the-art methods

We give the comparison of our CMB-Net versus four state-of-the-art methods in Table 6 and Fig. 8. The results of the three indicators of sensitivity, specificity and accuracy evaluated by the methods of SAE [11], SLFN + LReLU [10] and SAR-DNN [30] using ten-fold cross-validation were all significantly lower than the scores of CNN [9], which were about 4–6%.

We mainly compared them with the CNN method. The CNN method achieved the best sensitivity of 99.74%, but our CMB-Net yielded 98.18% accuracy which is marginally worse. In the evaluation results of specificity indicator, our method is about 2% higher than CNN score. In the comparison of accuracy score, our method looks similar to CNN's method, but it is still slightly higher than the CNN method. The high

Table 6 Comparison with state-of-the-art methods

Method	Sensitivity	Specificity	Accuracy	Validation
SAE [11]	93.20%	93.25%	93.22%	10-fold cross-validation
SLFN + LReLU [10]	93.05%	93.06%	93.06%	10-fold cross-validation
SAR-DNN [30]	95.13%	93.33%	94.23%	10-fold cross-validation
CNN [9]	99.74%	96.89%	98.32%	Hold-out validation
CMB-Net (ours)	98.18%	98.60%	98.39%	5-fold cross-validation

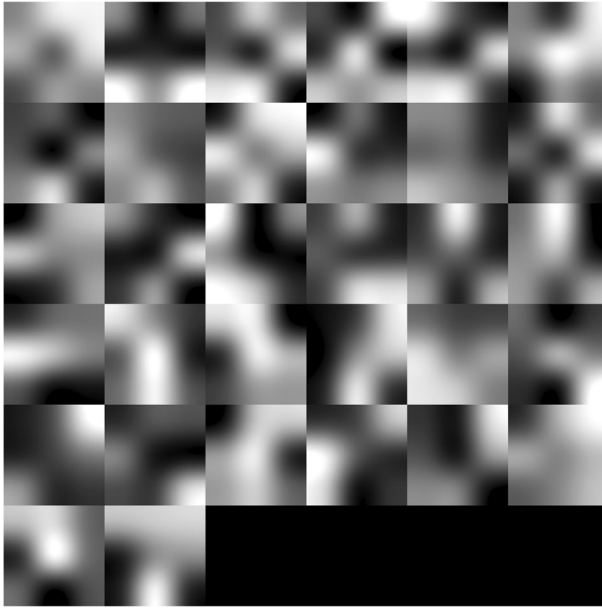


Fig. 8 Weights learnt from the CMB dataset

performance of our CMB-Net is contributed by the introduction of batch normalization as well as dropout regularization.

Overall, our CMB-Net is the best in the list. Moreover, our results were obtained by five-fold cross-validation while CNN was evaluated by simple hold-out validation. It can be concluded that our CMB-Net has achieved state-of-the-art classification performance in detecting CMB Fig. 9.

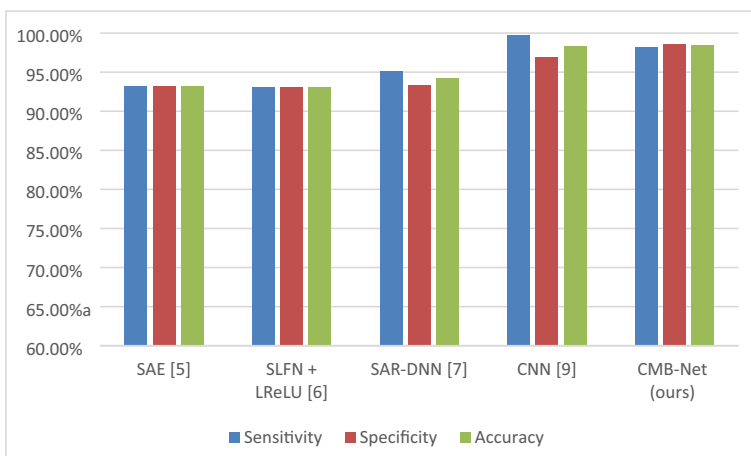


Fig. 9 Comparison with state-of-the-art methods

5 Conclusion

In this paper, a novel CMB classification method is proposed based on a deep CNN model named CMB-Net. The 18-layer CMB-Net was trained by RMSProp and the performance evaluation was implemented on a dataset of 13,031 samples based on five-fold cross-validation. The average classification accuracy was 98.39%, which revealed that our CMB-Net is accurate and robust for CMB classification. Moreover, the total running time of the five-fold cross-validation was only 184.79 s, which is really fast.

Nevertheless, it is hard to interpret the parameters in the CMB-Net, and we don't know why the network classified a certain sample as CMB or non-CMB. Additionally, we only solved a binary classification problem, we will extend our research to multiple class classification, such as multiple rating of the samples.

Acknowledgments The paper is supported by Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD), British Heart Foundation Accelerator Award, UK; Fundamental Research Funds for the Central Universities (CDLS-2020-03); Key Laboratory of Child Development and Learning Science (Southeast University), Ministry of Education.

References

1. Barnes SR, Haacke EM, Ayaz M, Boikov AS, Kirsch W, Kido D (2011) Semiautomated detection of cerebral microbleeds in magnetic resonance images. *Magn Reson Imaging* 29(6):844–852
2. Bian W, Hess CP, Chang SM, Nelson SJ, Lupo JM (2013) Computer-aided detection of radiation-induced cerebral microbleeds on susceptibility-weighted MR images. *Neuroimage Clin* 2:282–290
3. Chen Y, Villanueva-Meyer JE, Morrison MA, Lupo JM (2018) Toward automatic detection of radiation-induced cerebral microbleeds using a 3D deep residual network. *J Digit Imaging* 32:766–772
4. Fazlollahi A, Meriaudeau F, Giancardo L, Villemagne VL, Rowe CC, Yates P, Salvado O, Bourgeat P, AIBL Research Group (2015) Computer-aided detection of cerebral microbleeds in susceptibility-weighted imaging. *Comput Med Imaging Graph* 46:269–276
5. Govindaraj VV (2019) High performance multiple sclerosis classification by data augmentation and AlexNet transfer learning model. *J Med Imaging Health Inf* 9(9):2012–2021
6. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition, presented at the the IEEE conference on computer vision and pattern recognition (CVPR)
7. Hinz P, van de Geer S (2019) A framework for the construction of upper bounds on the number of affine linear regions of ReLU feed-forward neural networks, (in English). *IEEE Trans Inf Theory* 65(11):7304–7324
8. Hong J (2019) Detecting cerebral microbleeds with transfer learning. *Mach Vis Appl* 30(7–8):1123–1133
9. Hong J (2020) Classification of cerebral microbleeds based on fully-optimized convolutional neural network. *Multimed Tools Appl* 79:15151–15169
10. Hou X-X (2018) Voxelwise detection of cerebral microbleed in CADASIL patients by leaky rectified linear unit and early stopping. *Multimed Tools Appl* 77(17):21825–21845
11. Hou X-X, Chen H (2016) Sparse Autoencoder based deep neural network for voxelwise detection of cerebral microbleed. In: 22nd International Conference on Parallel and Distributed Systems, Wuhan, China, pp. 1229–1232: IEEE
12. Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K (2016) SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size, "arXiv:1602.07360
13. Jiang Y (2018) Exploring a smart pathological brain detection method on pseudo Zernike moment. *Multimed Tools Appl* 77(17):22589–22604
14. Krizhevsky A, Sutskever I, Hinton G (2012) ImageNet classification with deep convolutional neural networks. *Int Conf Neural Inf Process Syst*, pp. 1097–1105
15. Kuijf HJ et al (2012) Efficient detection of cerebral microbleeds on 7.0 T MR images using the radial symmetry transform. *Neuroimage* 59(3):2266–2273

16. Momeny M, Jahanbakhshi A, Jafarnejad K (2020) Accurate classification of cherry fruit using deep CNN based on hybrid pooling approach. *Postharvest Biol Technol* 166:9 Art. no. 111204, (in English)
17. Naggaz N (2009) Remote-sensing image classification based on an improved probabilistic neural network. *Sensors* 9(9):7516–7539
18. Pan C (2018) Multiple sclerosis identification by convolutional neural network with dropout and parametric ReLU. *J Comput Sci* 28:1–10
19. Pan C (2018) Abnormal breast identification by nine-layer convolutional neural network with parametric rectified linear unit and rank-based stochastic pooling. *J Comput Sci* 27:57–68
20. Qian P (2018) Cat swarm optimization applied to alcohol use disorder identification. *Multimed Tools Appl* 77(17):22875–22896
21. Reddy RVK, Rao BS, Raju P (2018) Handwritten hindi digits recognition using convolutional neural network with rmsprop optimization (2nd international conference on intelligent computing and control systems). New York: IEEE, pp. 45–51
22. Ruder S (2016) An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*
23. Simonyan K, Zisserman A (2015) Very DEEP convolutional networks for large-scale image recognition. *Int Conf Learn Represent*
24. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. *J Mach Learn Res* 15(1):1929–1958
25. Sun J (2018) Preliminary study on angiosperm genus classification by weight decay and combination of most abundant color index with fractional Fourier entropy. *Multimed Tools Appl* 77(17):22671–22688
26. Tang C (2018) Twelve-layer deep convolutional neural network with stochastic pooling for tea category classification on GPU platform. *Multimed Tools Appl* 77(17):22821–22839
27. Wu LN (2008) Improved image filter based on SPCNN, (in English). *Sci China Ser F Life Sci* 51(12):2115–2125
28. Wu LN (2008) Pattern recognition via PCNN and Tsallis entropy," (in English). *Sensors* 8(11):7518–7529
29. Wu LN (2009) Segment-based coding of color images," (in English). *Sci China Ser F Life Sci* 52(6):914–925
30. Zhang Y-D, Zhang Y, Hou X-X, Chen H, Wang S-H (2017) Seven-layer deep neural network based on sparse autoencoder for voxelwise detection of cerebral microbleed. *Multimed Tools Appl* 77(9):10521–10538
31. Zhao G (2018) Smart pathological brain detection by synthetic minority oversampling technique, extreme learning machine, and Jaya algorithm. *Multimed Tools Appl* 77(17):22629–22648

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.