Machine Vision and Applications (2020) 31:36
https://doi.org/10.1007/s00138-020-01087-0

SPECIAL ISSUE PAPER

Using CNN with Bayesian optimization to identify cerebral micro-bleeds

Piyush Doke[1,2] · Dhiraj Shrivastava[3] · Chichun Pan[4] · Qinghua Zhou[5] · Yu-Dong Zhang[1,5,6]

Received: 9 September 2019 / Revised: 9 April 2020 / Accepted: 6 May 2020 / Published online: 30 May 2020

Abstract

This article studies the problem of detecting cerebral micro-bleeds (CMBs) using a convolutional neural network (CNN). Cerebral micro-bleeds (CMBs) are increasingly recognized neuroimaging findings, occurring with cerebrovascular diseases, dementia, and normal aging. Naturally enough, it becomes necessary to detect CMBs in the early stages of life. The focus of this article is to infuse new techniques like Bayesian optimization to find the optimum set of hyper-parameters efficiently, making even the simplest of CNN architectures perform well on the problem. Experimentally, we observe our CNN (five layers, i.e., two convolution, two pooling, and one fully connected) achieves accuracy = 98.97%, sensitivity = 99.66%, specificity = 98.14%, and precision = 98.54% on the test set (hold-out validation) when calculated over an average of ten runs. The proposed model outperformed state-of-the-art methods.

Keywords Cerebral micro-bleeding · CNN · Convolution filters · ReLU · Softmax · Max pooling · Batch normalization · Bayesian optimization · Gaussian process regression · Acquisition function · Image augmentation

1 Introduction

Cerebral micro-bleeds (CMBs) are increasingly recognized neuroimaging findings, emerging as diagnostic markers for cognitive impairment and dementia [1], stroke and intracerebral hemorrhages (ICH) [2], and cerebral amyloid angiopathy (CAA) [3]. Recent studies suggest that common etiologies can cause CMBs, including blood pressure, aneurysm, blood vessel abnormalities, blood disorders, head trauma, and brain tumors [4]. At the same time, some special etiologies that cause CMBs include cocaine abuse, posterior reversible encephalopathy, brain radiation therapy, intravascular lymphomatosis (IVL), thrombotic thrombocytopenic purpura (TTP), moyamoya disease, infective endocarditis (IE), sickle cell anemia, β-thalassemia, proliferating angioendotheliomatosis, cerebral autosomal dominant arteriopathy with subcortical infarcts and leukoencephalopathy (CADASIL), genetic syndromes, and obstructive sleep apnea (OSA) [5]. CMBs are tiny deposits of blood degradation products, mostly consisting of hemosiderin. Hemosiderin is a strong paramagnetic material and, hence, can be detected

Piyush Doke, Dhiraj Shrivastava, Chichun Pan and Qinghua Zhou have contributed equally to this work.

Basic Research Program of Jiangsu Province (BK20150983); National Key Research and Development Plan (2017YFB1103202); Henan Key Research and Development Project (182102310629); Royal Society International Exchanges Cost Share Award, UK (RP202G0230); Medical Research Council Confidence in Concept (MRC-CIC) Award; Medical Research Council Confidence in Concept Award, UK (MC_PC_17171); Hope Foundation for Cancer Research, UK (RM60G0680). Fundamental Research Funds for the Central Universities (CDLS-2020-03) and Key Laboratory of Child Development and Learning Science (Southeast University), Ministry of Education.

✉ Yu-Dong Zhang
yudongzhang@ieee.org

1 School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo, Henan, China

2 School of Mathematics and Computer Science, Indian Institute of Technology Goa, Veling, Goa, India

3 Department of Mechanical Engineering, Indian Institute of Technology Varanasi, Varanasi, Uttar Pradesh, India

4 School of Business, Nanjing Normal University, Nanjing, Jiangsu, China

5 Department of Informatics, University of Leicester, Leicester, Leicestershire, United Kingdom

6 Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

with a magnetic field [6]. This phenomenon, known as the susceptibility effect, forms the basis for CMB imaging techniques like three-dimensional T2*-GRE [7] and susceptibility-weighted imaging (SWI) [8], with SWI being the most sensitive one to date.

As the frequency of CMBs varies enormously depending on the MRI study characteristics and selection of the study subjects, reported prevalence in different clinical conditions has considerably wide ranges: 18% to 71% in ischemic stroke [9,10], 47% to 80% in ICH [9,11], or 17% to 46% in cognitive decline/dementia [12]. Naturally enough, it becomes necessary to detect CMBs in the early stages of life. However, manual detection of CMBs is time-consuming, less accurate, and subjective, which is especially evident from the high inter-observer and intra-observer variability. These intricacies arise due to their complex morphological structure and widespread distribution throughout the brain [13]. Moreover, it is easy to miss smaller CMBs or even mistake them for vein cross sections as their sizes commonly range from 2 to 10 mm [14].

The current literature points to the fact that deep learning methods prove especially effective. Hence, we propose using a convolutional neural network (CNN) to detect CMBs. This method no longer demands the traditional handcrafted features and instead learns the features that are most relevant for making correct predictions, all by itself. However, a common limitation is the lack of exploration in the hyper-parameter space. This is evident by the previous models that fail to acquire a near-perfect accuracy, which is unneglectable in a clinical setting. Hence, we suggest using Bayesian optimization, a reasoning-based approach to find the perfect hyper-parameter sleast evaluations possible, as shown in Sect. 4. The integration of Bayesian optimization improves upon the traditional hit-and-trail methods like random search and grid search in both accuracy and training-efficiency perspectives. A comparison of this is illustrated in Sect. 5.6. We also employ image augmentation to introduce a regularization effect, therefore reducing over-fitting. Image augmentation helps increase the test set performance by generalizing to the data well, as illustrated in Sect. 5.3. All this results in a highly compact, efficient, and near-perfect classifier. In Sect. 5.7, we compare our model to some state-of-the-art methods. Again here, we see that our model scores better than the topmost performers, although being the smallest one (5 layers), especially when compared to ResNet-50 (50 layers) and DenseNet-201 (201 layers).

2 Related work

Many researchers have made efforts to solve the problem of CMB detection. Barnes et al [15] came up with a statistical thresholding algorithm to identify hypo-intensities within the images and employed support vector machines (SVM) to separate affirmative CMBs from the marked hypo-intensities based on features such as signal intensities and shapes. Bian et al. [16] developed and tested a semi-automated method for identifying CMBs on minimum intensity projected susceptibility-weighted MR images. Their algorithm utilized 2D fast radial symmetry transform initially to detect putative CMBs and then eliminated false positives by examining geometric features measured after performing 3D region growing on the potential CMB candidates. Fazlollahi et al. [17,18] presented two pieces of research. One utilizes a novel cascade of random forest classifiers that are trained on robust Radon-based features with an unbalanced sample distribution. Second, being a two-step technique that first detects and bounds the potential CMB candidates using multi-scale Laplacian of Gaussian. Then, inside each such bounding box, a set of robust three-dimensional Radon- and Hessian-based shape descriptors are extracted to train a cascade of binary random forests (RF). Chen et al. [19] proposed their method in a three-step approach: candidate localization with statistical thresholding, hierarchical 3D feature representation with deep CNN and SVM classification to reduce false positives. Van den Heuvel et al. [20] proposed a two-step method. In the first step, each voxel is characterized by 12 features based on the dark and spherical nature of CMBs, and a random forest classifier is used to identify candidate CMB locations. In the second step, segmentations are made from each identified candidate location. Subsequently, an object-based classifier is used to remove false-positive detections of the voxel classifier. Kaaouana et al. [21] gave a rather interesting method. They demonstrated a fast 2D phase processing technique for computing internal field maps (IFM), which makes it possible to characterize CMBs through their magnetic signature in a routine clinical setting, based on 2D multi-slice acquisitions. Wang et al [22] came up with a CNN-based approach for identifying CMBs that exploits rank-based average pooling scheme. Another innovative approach by Hong and Lu [23] enables CMB detection via discrete wave transformation and backpropagation neural network. A more recent method by Liu J. et al [24] promotes CMB detection using ResNet-50 with transfer learning to compensate for the limited number of training samples. At the same time, another transfer learning method is demonstrated by Tang C. et al [25] that utilizes DenseNet-201 as the basic algorithm.

3 Method

3.1 Feature learning through convolution

In the classification of elementary binary images, a simple feed forward neural network might suffice, but these networks perform poorly on images which have high spa-

Table 1 Summary of the model

Type of layer	Kernel size	# of Filters	Stride	Padding	# of Parameters	Feature map
Input_Image						(41, 41, 1)
Conv_2D_1	(3, 3, 1)	64	(1, 1)	(0, 0)	640	(39, 39, 64)
Batch_Norm_1					256	(39, 39, 64)
ReLU_1					0	(39, 39, 64)
Max_Pool_2D_1	(4, 4, 1)		(4, 4)	(0, 0)	0	(9, 9, 64)
Conv_2D_2	(3, 3, 64)	128	(1, 1)	(1, 1)	73856	(9, 9, 128)
Batch_Norm_2					512	(9, 9, 128)
ReLU_2					0	(9, 9, 128)
Max_Pool_2D_2	(4, 4, 1)		(4, 4)	(0, 0)	0	(2, 2, 128)
FC_1					1026	(1, 1, 2)
Softmax_1					0	(1, 1, 2)

Total parameters: 76,290
Trainable parameters: 75,906
Non-trainable parameters: 384
Note: Dropout is not used in this model

tial dependences. Nevertheless, convolutional neural networks are successful in capturing these spatial and temporal dependencies through the application of relevant filters. An additional benefit of using convolution is its reusability of weights, which leads to a reduction in trainable parameters. Owing to these reasons, we adopt convolution into our approach. The convolution operation consists of two main components:

(a) Feature Maps: The CNN performs a series of convolution operations. During each of such operations, the inputs, commonly known as input feature maps, are converted into output feature maps using filters. In our case, inputs to the CNN will be grey-scale images which serve as the initial input feature maps. A grey-scale image consists of a 2D matrix, where each element of the matrix represents the intensity of the corresponding pixel, hence making our initial input feature maps 2D in shape.

(b) Filters: A filter is a 3D weight matrix, whose height and width are hyper-parameters defined by the user and whose depth is equal to that of the input feature map. Using filters is a common technique in image processing for enhancing any given image. It enables us to emphasize certain features of the image or even remove other features. For a deep learning model, this is especially vital, as it can help bring out the relevant features of the image and train on them alone while neglecting the redundant ones, which is conducive to better classification. It depends on the application as to what filters might work the best. Hence, the filters we use in our CNN are matrices with variables as its elements, usually referred to as weights. These weights are learned by the CNN itself, based on the data fed to the network. This way, we let the model select the features it wants to train on, ensuring maximum precision.

During the process of convolution, a filter is stridden over the input feature map, performing element-wise multiplication with the portion of the feature map it is currently on, and then summing up the results into a single pixel. The filter repeats this process for every location it strides over, creating an output feature map of a 2D shape. Finally, there is the bias term unique to every filter matrix, which is used to perform an element-wise sum with the resultant matrices. The use of multiple filters can result in multiple 2D output matrices. Hence the operation is extended in such a case by stacking the output matrices over each other, giving rise to 3D output feature maps.

This operation can be seen in Fig. 1. The filters start from the top-left corner of the input feature map and move to the right with a specific stride value until they parse the complete width. After that, they hop down to the extreme left with the same stride value and repeat this process until the entire feature map is traversed. The convolution operation can be performed in two ways:

- **(a) Valid Padding:** The dimensions of the convoluted image are less than those of the input image. This method needs no prior preparation as a reduction in dimensionality is a natural consequence of convolution.
- **(b) Same Padding:** The dimensions of the resultant image are the same as those of the input image. This method can be achieved by padding the input image with a border of zero-value pixels. So when this padded image is convoluted, the height and width fall back to the original dimensions. The general formula for finding the height of the image after a convolution operation is given in Eq. (1), where H is the height of the image, W is the width, s is the value of stride, p is the padding, and $[l]$ denotes the l^{th} layer.

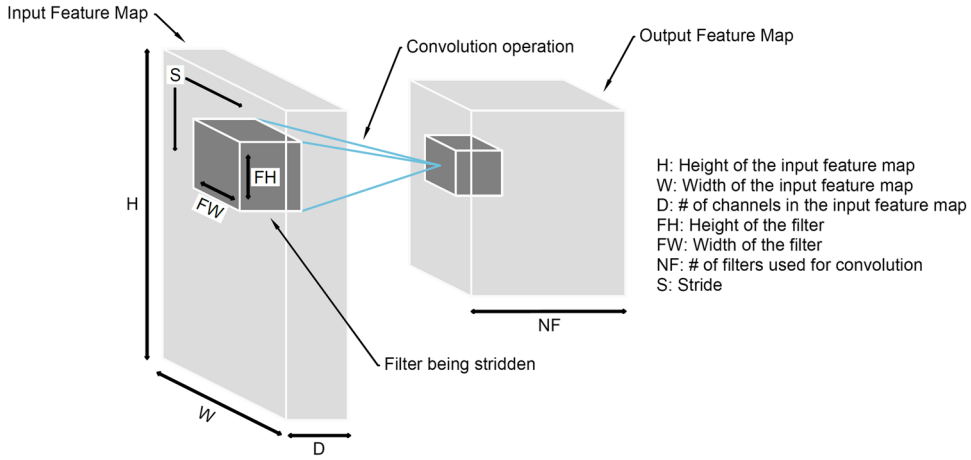

Fig. 1 Convolution operation

$$n_{\mathrm{H/W}}^{[l]} = \left\lfloor \frac{n_{\mathrm{H/W}}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} \right\rfloor + 1. \tag{1}$$

3.2 Structure of the CNN

The model consists of two convolutional layers and one dense layer. The input image is fed into the first convolutional layer, which consists of 64 filters, each of kernel sizes 3*3*1. The output of this convolutional layer is batch normalized and passed to a ReLU activation function, whose output is further passed to a max-pooling layer with a kernel size of 4*4*1. A similar feature learning block consisting of a convolutional layer, batch normalization, ReLU activation, and max-pooling is then applied. All kernel sizes are the same as the previous feature learning block, while a total of 128 filters are used in the convolutional layer. The output of our feature learning layers is then flattened and given to a dense layer, which is coupled to a softmax activation function. This classifies the samples into CMBs and Non-CMBs. A summary of the model is given in Table 1.

3.3 Training

After building the model as specified in Sect. 3.2, we proceed to train it. Out of several different training algorithms, we specifically choose Adam [26] (adaptive moment estimation). Our choice allows for better learning due to the combined effect of momentum and scaling, aspects of SGD with momentum and RMS-prop, respectively.

Momentum attributes to faster learning, as it takes the moving average of the past gradients. With this characteristic, SGD helps produce gradients that are consistent in the direction of the optimum. RMS-prop allows for scaling the gradients. It first calculates the moving average of squared gradients and then scales the gradients found using momentum to damp out undesired oscillations.

$$v_{\mathrm{dp}}^{t} = \beta_1 * v_{\mathrm{dp}}^{t-1} + (1 - \beta_1) * \mathrm{dp}^{t} \tag{2}$$

$$s_{\mathrm{dp}}^{t} = \beta_2 * s_{\mathrm{dp}}^{t-1} + (1 - \beta_2) * (\mathrm{dp}^{t})^2 \tag{3}$$

$$\hat{v}_{\mathrm{dp}}^{t} = \frac{v_{\mathrm{dp}}^{t}}{1 - (\beta_1)^t} \tag{4}$$

$$\hat{s}_{\mathrm{dp}}^{t} = \frac{s_{\mathrm{dp}}^{t}}{1 - (\beta_2)^t} \tag{5}$$

$$p = p - \alpha * \frac{\hat{v}_{\mathrm{dp}}^{t}}{\sqrt{\hat{s}_{\mathrm{dp}}^{t}} + \epsilon}. \tag{6}$$

Equations (2) to (6) represent the mathematical implementation of these two Adam components. Here, β_1 is the momentum term, β_2 is the RMP-prop term, p is some parameter of the network, dp is the derivative of the cross-entropy function w.r.t. p, t is the time, α is the learning rate, and ϵ is an epsilon term for avoiding division by zero. It should be noted that only in Eqs. (4) and (5), t is used as an exponent, i.e., $(\beta_i)^t$. All other occurrences of t should be treated as notations.

Due to the momentum and scaling effects, Adam is generally regarded as fairly robust when compared to the other optimizers, given the hyper-parameter choices are ideal. These aspects of Adam lead us toward tuning the model for yielding maximum possible accuracy. However, tuning can be a lengthy process due to the limitless combinations of hyper-parameters that can be tested. Hence, we go for a technique that combines search with reasoning for greater efficiency, namely Bayesian optimization.

4 Tuning hyper-parameters using Bayesian optimization

Hyper-parameter tuning aims to find the right hyper-parameter choices for a given machine learning algorithm so that it returns the best performance when evaluated on a test set. As shown in Eq. (7), where f is considered to be the performance, x is some hyper-parameter setting, and x_{opt} is the optimum choice.

$$x_{\text{opt}} = \underset{x \in \mathcal{X}}{\text{argmax}}\, f(x). \tag{7}$$

There several ways to achieve this, with grid and random search being the most common ones. These methods are suitable for hyper-parameter tuning, but they do not learn anything from the evaluated hyper-parameter sets during the tuning process. This disadvantage of grid and random search methods is why we turn to Bayesian optimization.

Bayesian optimization builds a probability model that maps hyper-parameter values to the probability of getting a certain value of the objective function, often referred to as the score. Using these probabilities, it selects the most promising hyper-parameter values to evaluate the true objective function.

While tuning hyper-parameters, the main problem arises when an objective function is too expensive to compute. Because for each set of new hyper-parameters, we have to train the model from scratch to find out how well it performs. At the same time, calculating gradients for most of the hyper-parameters is impossible, leaving re-iteration the only option. So there is a need for a cheaper probability model that approximates the true objective function, and we try to achieve the same using Bayesian optimization. Bayesian optimization is a class of mathematical methods for optimizing expensive functions. These methods start by building a Bayesian statistical model (also called the "surrogate") on the objective function and are represented by $P(y|x)$, where y is the score and x is some set of the hyper-parameters. In most cases, this statistical model is a Gaussian process prior.

The process of Bayesian optimization is as follows:

(a) Build a Gaussian process prior model of our objective function: Gaussian process models are generally good for predicting results of future experiments and hence serve well for our purpose. They assume that similar inputs give a similar output, which means that the objective function is smooth. As we do not know much about the hyper-parameters, this prior is a sensible assumption. These models also learn the appropriate scale for measuring similarity, i.e., they can realize the scale of each hyper-parameter setting. This scale can indicate the amount of difference needed to expect very different results.

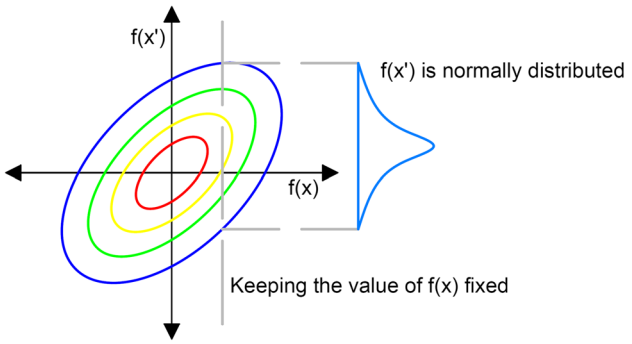

Fig. 2 Conditioning on a multivariate Gaussian

Gaussian processes predict a distribution, instead of a single value, for every hyper-parameter setting. They are called Gaussian processes because these predictions are Gaussian distributed. For predictions that are close to several consistent training cases, the predicted Gaussian curves are relatively sharp and have less variance. However, for predictions that lie far away, curves tend to be more spread out and exhibit high variance.

Mathematically, Gaussian process regression can be understood as follows. Say we have a function "f" that we wish to model. Given x and x', we put the corresponding function outputs, i.e., $f(x)$ and $f(x')$, in a vector and assume it to be drawn from a multivariate Gaussian, as shown in Eq. (8). Here, we have $\sum$ (kernel) a function that decreases with $||x - x'||$ and μ as another function, which is usually set to a constant.

$$\begin{bmatrix} f(x) \\ f(x') \end{bmatrix} \sim N\left(\begin{bmatrix} \mu(x) \\ \mu(x') \end{bmatrix}, \begin{bmatrix} \sum(x,x) & \sum(x,x') \\ \sum(x',x) & \sum(x',x') \end{bmatrix} \right). \tag{8}$$

Figure 2 shows this Gaussian distribution in picture. If we take the conditional probability keeping $f(x)$ fixed, we observe that $f(x')$ is normally distributed.

Gaussian regression takes this same calculation and generalizes it to multiple dimensions.

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_k) \end{bmatrix} \sim N\left(\begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_k) \end{bmatrix}, \begin{bmatrix} \sum(x_1,x_1) & \dots & \sum(x_1,x_k) \\ \vdots & \ddots & \vdots \\ \sum(x_k,x_1) & \dots & \sum(x_k,x_k) \end{bmatrix} \right). \tag{9}$$

We then proceed to Bayesian linear regression using all the first $k-1$ observations, to analytically compute the posterior on a new k^{th} point, given the rest of the observations. This is achieved as follows:

$$\begin{aligned} \mu_k &= \sum(x_k, x_{1:k-1}) \sum(x_{1:k-1}, x_{1:k-1})^{-1} f(x_{1:k-1}) \\ \sigma_k^2 &= \sum(x_k, x_k) - \sum(x_k, x_{1:k-1}) \end{aligned} \tag{10}$$

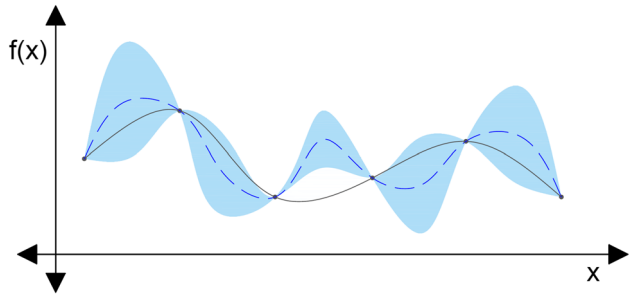

Fig. 3 Gaussian process regression

$$\times \sum (x_{1:k-1}, x_{1:k-1})^{-1} \sum (x_{1:k-1}, x_k). \quad (11)$$

Using the formulas shown in Eqs. (10) and (11), we are now capable of computing the distribution at any required point x_k, whose mean is μ_k and variance is σ_k^2. In effect, if we take a large (infinitely many) number of points, we can draw a whole curve passing through their means. In addition, we can also have the confidence intervals around them, hence forming our Bayesian statistical model on the objective function, as shown in Fig. 3. The black curve in the figure is the true objective function, and the blue dashed curve is the Gaussian regression model with confidence interval in sky-blue. Points in black are the already explored sets hyper-parameters.

(b) Find the hyper-parameter values that maximize the acquisition function: A good strategy about which setting to try next can be as follows. We can keep track of the best setting so far while evaluating settings in the region where it is valuable to learn (where the acquisition function is optimum).

Say we want to maximize our objective function, and we model the same using a Gaussian process, as in Fig. 4. Now, consider the three predicted distributions on points a, b, and c for three different hyper-parameter settings we would like to try next. The dashed blue line represents the mean. Notice that the upper limit of the confidence interval at b and c is less than that at a. So it is rational to try and evaluate the objective function at a. This is because we are quite uncertain about that region, and it is possible to find a new maximum at point a. So learning at a is more valuable. We, therefore, require a function whose optimum corresponds to this point a and is cheaper to evaluate and optimizable using our traditional methods of gradients and Hessians. This function is often known as the expected improvement acquisition function.

Mathematically, we implement it as follows. If we have a posterior function f, which models the loss, and after n evaluations, we find the minimum of this posterior to be f^* for some x^*. If we do one more evaluation, our posterior gets updated, and the value of the objective function at some new point x is revealed to us, say $f(x)$. If we wish to stop here, the solution to our optimization problem would be $\min(f^*, f(x))$. So the reduction in loss now becomes the expected difference between f^* and $\min(f^*, f(x))$, which

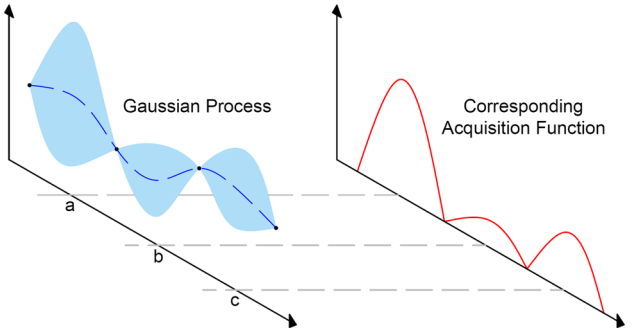

Fig. 4 Acquisition function for choosing the next point of evaluation

is conditioned over the n-times evaluated posterior, as shown in Eq. (12).

$$\text{Expected Improvement} = E_n[f^* - \min(f^*, f(x))]. \quad (12)$$

Equation (12) can be further converted into Equation (13), where n signifies that n evaluations have already been done, f^* is the optimum after first n evaluations, μ_n is the mean, σ_n is the standard deviation, φ is the probability density function of a normal, and Φ is the cumulative density function of a normal. Optimally, we evaluate the objective at a point that gives the maximum expected improvement.

$$\begin{aligned}&\text{Expected Improvement}\\ &= [f^* - \mu_n(x)]^+ + \sigma_n(x)\varphi\left(\frac{f^* - \mu_n(x)}{\sigma_n(x)}\right)\\ &\quad -|f^* - \mu_n(x)|\Phi\left(-\frac{|f^* - \mu_n(x)|}{\sigma_n(x)}\right). \quad (13)\end{aligned}$$

(c) Evaluate the objective function and incorporate the results into the Gaussian process posterior: Our next step requires us to evaluate the objective function using the set of hyper-parameters obtained previously via acquisition function. This new observation leads to an improved understanding of the objective we aim to model and moves our Gaussian process posterior from n evaluations to $n + 1$. The same is depicted in Fig. 5, where x is the hyper-parameter setting suggested by the acquisition function.

(d) Repeat steps (b) and (c) until the maximum number of iterations is reached.

5 Experiments

5.1 Subjects

The CMB samples used in this article were collected from patients with cerebral autosomal dominant arteriopathy with subcortical infarcts and leukoencephalopathy (CADASIL), while the otherwise samples were procured from some

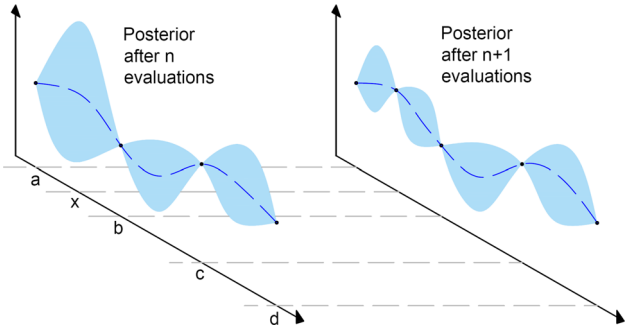

Fig. 5 Gaussian process regression

Table 2 Background information about the subjects

Criteria	CADASIL	HC
Number of subjects	10	10
Age (years)	51.6 ± 9.6	53.8 ± 6.1
Gender (male/female)	14/16	14/16
Education level (years)	12.2 ± 2.1	11.9 ± 2.8
Disease duration (years)	17.9 ± 17.1	–

healthy volunteers, also referred to as healthy controls (HCs). Their 3D volumetric brain images of dimensions 364*448*48 were reconstructed using the Syngo MR B17 software. More background information about the subjects is given in Table 2.

Doctors specializing in neuroradiology were employed to mark CMB voxels from the image data manually. Voxels were marked CMBs if they were labeled "possible" or "affirmative," while other classes were treated as non-CMBs. Any conflicting samples were judged based on votes. The exclusion criteria were as follows: (1) Blood vessels were discarded and (2) lesions larger than 10mm were discarded.

5.2 Data generation

5.2.1 Image preprocessing

Input dataset is generated using a sliding window of 41*41 pixels. First, we slice the main 3D image into 2D images, hence resulting in multiple cross-sectional axial plane image slices of the brain varying in depth. The sliding window operation is then used to fragment these sliced images into pieces of size 41*41 pixels. The window size of 41*41 is specifically chosen as it provides optimally sized images (in terms of memory) without compromising much on the quality, leading to higher efficiency in training. The sliding window is stridden over the image from left to right and top to bottom to produce samples. As for labeling, we check if the central pixel p of the sample of interest is a corresponding CMB voxel in its 3D counterpart or not, as shown in Eq. (14). Figure 6 shows these labeled samples.

Table 3 Types of augmentations used

Augmentation	Mean	Variance	Angle
Gaussian noise	0	0.1*255	–
Gaussian blur	0	1	–
Rotation	–	–	(-25,25)

$$\text{label} = \begin{cases} \text{true (1)}, & \text{if p belongs to CMB} \\ \text{false (0)}, & \text{otherwise.} \end{cases} \tag{14}$$

5.2.2 Train and test set

The dataset contains a total of 13031 samples, out of which 6407 belong to CMBs and 6624 correspond to non-CMBs. We implement hold-out validation by splitting the data randomly into two groups, namely training set and test set. The split fraction is 0.7, i.e., 70% of the samples are put into the training set, while the remaining 30% are used for evaluating the deep learning model.

Moreover, data that are reserved for training undergo image augmentation. Each sample in the training set is augmented to produce a new artificial image. The original and augmented sets are then combined to form the new training set, doubling the size of the training samples we have. Test data remain unaugmented.

5.3 Image augmentation

To make our deep learning model perform better, we use image augmentation. This technique produces artificial images by combining multiple augmentations to reduce overfitting and therefore improving accuracy on the test set. The types of augmentations we use in our application are shown in Table 3. As per our observations, only one augmentation from blur and noise is chosen at a time, as they tend to cancel out each other. We illustrate the augmented images in Fig. 7.

5.4 Results obtained by our proposed method

To evaluate the performance of our model, we use the measure of accuracy. Accuracy tells us how close our predictions are to the ground truth. We also employ some extra performance measures for future experiments.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{15}$$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{16}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{17}$$

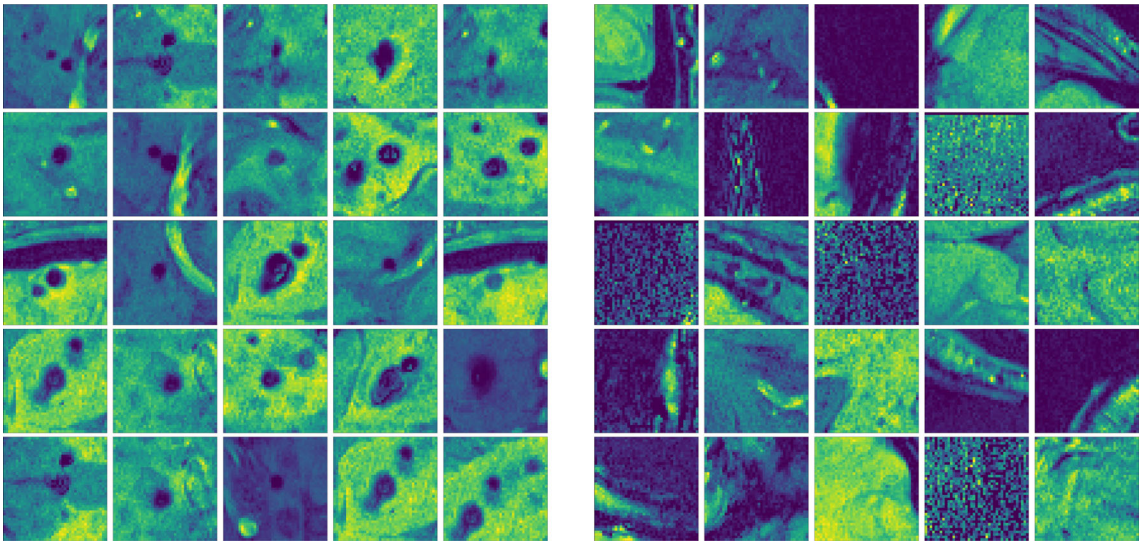

Fig. 6 The left panel of images represents the samples with CMBs, and the right panel of images represents the non-CMB samples

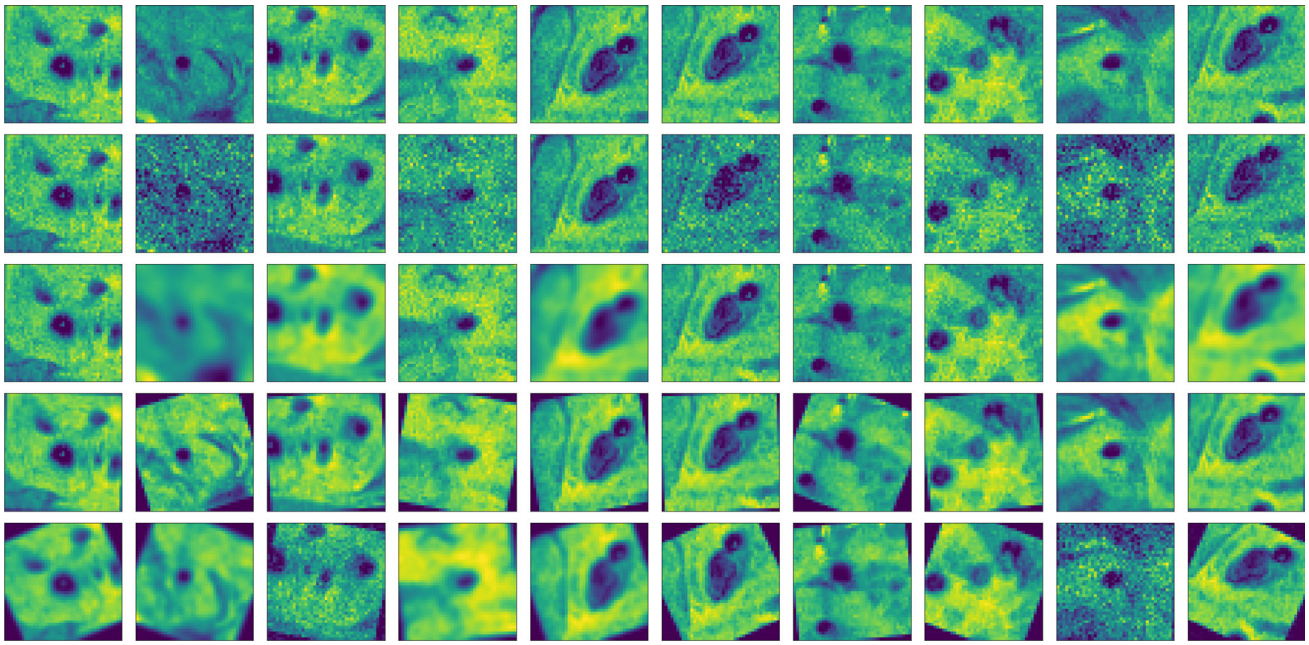

Fig. 7 The first row shows the original unaugmented images, the second row onward augmentations are applied in the sequence given in Table 3, and the last row corresponds to the mixed effect of all augmentations together

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{18}$$

All the metrics of interest are illustrated in Eqs. (15) to (18), where TP is true positive, TN is true negative, FP is false positive, and FN is false negative.

In this experiment, we illustrate the results of using Bayesian optimization. We choose three hyper-parameters to optimize—learning rate, momentum term, and the RMS-prop term. Detailed information on the range to probe into the hyper-parameter space, as well as the results of Bayesian optimization, is given in Table 4.

To evaluate performance, we proceed to train a model from scratch using the optimum values. Performance results are illustrated in Fig. 8 and Table 5. An accuracy of 99.73% was observed on the training set and accuracy of 98.21% on the test set. The model was trained for 5 epochs with a mini-batch size of 64 samples. The optimum hyper-parameter settings were found in just 13 iterations of Bayesian optimization.

Table 5 shows the accuracy on training and test sets before we proceed to train the model and after we are done training

Table 4 Hyper-parameter settings

Hyper-parameter name	Range to probe	Optimum value
Learning Rate	$(10^{-6}, 1)$	$10^{-2.703}$
Momentum Term	(0.0001, 0.9999)	$4.015 * 10^{-1}$
RMS-prop Term	(0.0001, 0.9999)	$9.999 * 10^{-1}$

Training algorithm used: Adam
Mini-batch size: 64
Number of epochs: 3
Optimum hyper-parameter settings found in iteration number: 13

it with the optimal set of hyper-parameters. It is evident from the table that both train and test accuracy starts from 39.45% and 28.24%, respectively. As shown in Fig. 8, due to a steep gain in the first iteration, we do not see a gradual increase. We also observe that training accuracy does a commendable job at staying high; contrastingly, test accuracy fluctuates in the second iteration. This fluctuation occurs due to the flow of gradients that promote over-fitting. The accuracy on the training set increases slightly, making the model memorize the training samples. However, starting again from the third iteration, training accuracy falls, and test accuracy rises. This boost in test performance can be justified by the noise we introduce in the training data using image augmentation, as described in Sect. 5.3.

5.5 Comparison of different CNN structures

Deciding upon the structure of CNN can be quite difficult since the perfect structure can vary from application to application. Hence, to find the best fit for our problem, we explore a few possible structures. Due to the size of images being 41*41, we restrict our model with a maximum of two convolutional layers, as two convolutional layers already reduce the dimensions of the original image to a feature map with size of 9*9, making another iteration of convolution and pooling impossible.

This experiment was performed by training six CNNs from scratch, each of which had its hyper-parameters tuned by Bayesian optimization. Adam was used as the optimization algorithm, with a mini-batch size of 64 and the number of epochs set to 5.

From the performances shown in Fig. 9, we observe that models with one convolutional layer perform weakly, as one such layer is not enough to extract the required features. Even after increasing the dense layers, depreciation is noticed. A plausible explanation for this is attributed to the inability of a single convolutional layer to bring out the best possible features to learn on. Instead, the single convolutional layer promotes learning on the redundant features, forcing the feed-forward network to learn on incorrect data, hence causing the poor test set performance. Adding more fully connected layers generally allows for a complex classifier, but compromises on the generalization. This effect is especially noticed in the decreasing test set performance with an increasing number of fully connected layers.

Adhering to these facts, we hypothesize that more convolution layers and less dense layers are required for better performance and therefore build three more models to test it. Continuing our experiment, we find that our hypothesis indeed holds.

Moving on to the models with additional convolution layers, we observe performance gains up to 99.73% on the training set and 98.28% on the test set. The accuracy on the training set remains approximately equal on all three models. However, when referring to the test performance, we see a slight dip in the model with two fully connected layers. Models with one and three fully connected layers perform similarly, with the latter having an almost negligible lead. Nevertheless, given that the performance is quite similar, we prefer the model with one fully connected layer as it is smaller and efficient.

5.6 Comparing Bayesian optimization with grid search and random search

In this experiment, we compare the performance of random and grid search to Bayesian optimization. Random search and grid search are some alternatives to Bayesian optimization. However, they tend toward being inefficient, as they do not learn anything from their past evaluations of the objective function.

We perform a maximum of 847 evaluations in a grid search. 847 is specifically chosen, as it lets us create a 7*11*11 grid, enabling us to perform a thorough grid search. To keep it fair, we iterate random search 847 times as well. Table 6 shows the details of the hyper-parameter space to probe for both these methods and also the results.

Using the optimal settings found by all the three methods, we trained three models from scratch and compared their performances side by side. Accuracy is considered the primary measure to evaluate performance, but we also employ the previously discussed secondary measures of sensitivity, specificity, and precision.

Table 7 and Fig. 10 show this head-to-head performance comparison. We see that the results are quite accurate in all three methods, and this is due to the structure of the network and the techniques used. The biggest difference is seen in the number of evaluations all three methods perform. As expected, Bayesian optimization takes just 13 iterations to find the optimum value, in comparison with 417 and 276 of grid search and random search, respectively. These results provide proof to our hypothesis, on why a reasoning-based approach is better than hit-and-trial methods like grid search and random search.

Table 5 Training results

	Before training (%)	After training (%)
Training set accuracy	39.45	99.73
Test set accuracy	28.24	98.21

Training algorithm used: Adam
Number of epochs: 5
Mini-batch size: 64

Table 6 Hyper-parameter settings

Grid search Hyper-parameter name	Range to probe	Optimum value
Learning rate	$(10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1)$	10^{-3}
Momentum term	(0.0001, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.9999)	$4 * 10^{-1}$
RMS-prop term	(0.0001, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.9999)	$9 * 10^{-1}$
Random search Hyper-parameter name	Range to probe	Optimum value
Learning rate	$(10^{-6}, 1)$	$10^{-2.831}$
Momentum term	(0.0001, 0.9999)	$3.136 * 10^{-1}$
RMS-prop term	(0.0001, 0.9999)	$7.717 * 10^{-1}$

Training algorithm used: Adam
Mini-batch size: 64
Number of epochs: 3
For grid search, optimum hyper-parameter settings found in iteration number: 417
For random search, optimum hyper-parameter settings found in iteration number: 276

5.7 Comparison with the state-of-the-art methods

We compared our model to some of the state-of-the art methods. These methods include four-layer SAE [27], seven-layer SAE [28], CNN+RAP [22], DWT+PCA+BPNN [23], eight-layer CNN [29], nine-layer CNN-SP [30], ResNet 50 [24], and DenseNet-201 [25]. Performance results are based on the four average measurements of ten runs.

Figure 11 shows this comparison plot. Our method performs better than the rest with an accuracy = 98.97%, sensitivity = 99.66%, specificity = 98.14%, and precision = 98.54%. DenseNet-201 [25] comes closest to our approach, but all of its metrics stop well before the 98% mark. ResNet 50 [24] performs better than our model in terms of specificity, but good performance in just one aspect makes the model quite unbalanced. While comparing our approach to both the above methods, we find them to be computationally heavy. Our model achieves a higher accuracy with just five layers instead of the 201 layers of DenseNet and 50 layers of ResNet. Even though these networks are enormous and can

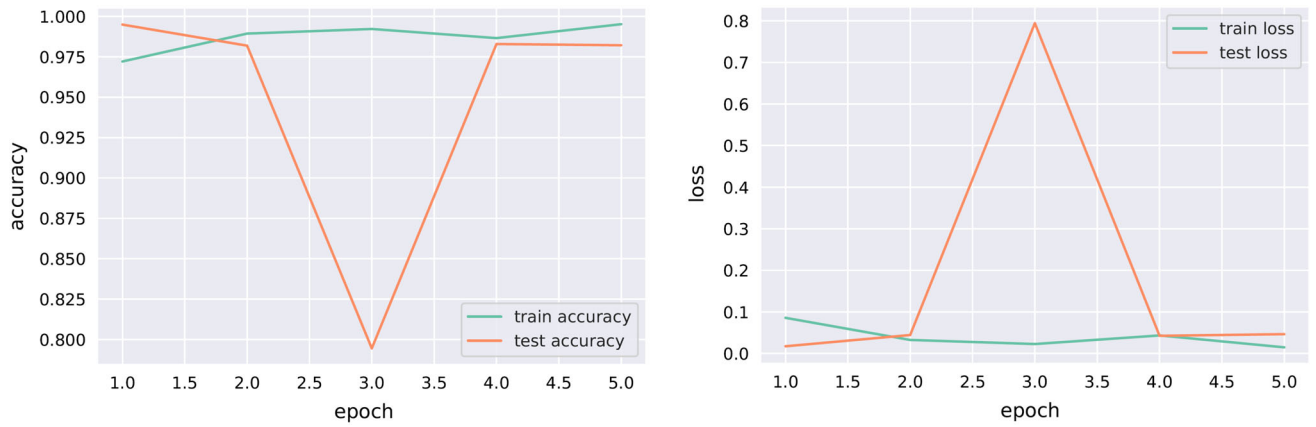

Fig. 8 Accuracy and Loss curves

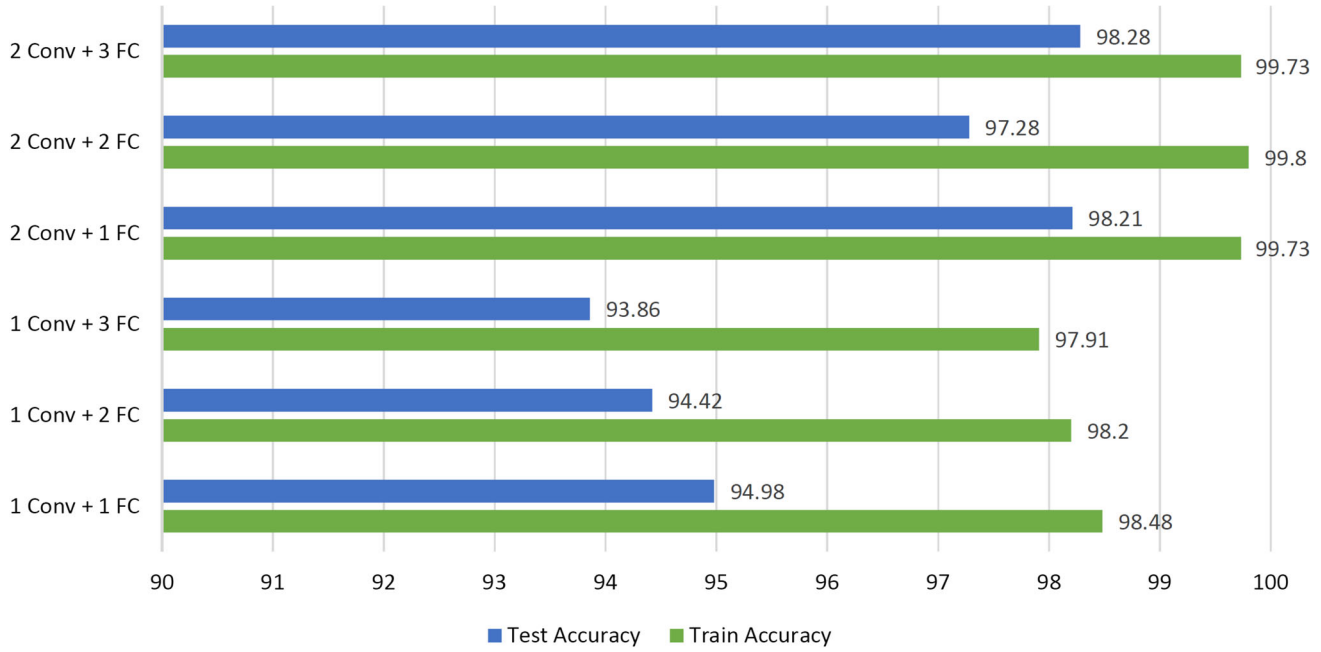

Fig. 9 Comparison of different CNN structures

Table 7 Training results

Method (%)	Accuracy (%)	Sensitivity (%)	Specificity (%)	Precision (%)	Iterations required
Bayesian optimization	98.21	97.11	99.55	99.62	13
Random search	99.51	99.67	99.32	99.44	417
Grid search	99.08	99.53	98.52	98.80	276

Training algorithm used: Adam
Number of epochs: 5
Mini-batch size: 64

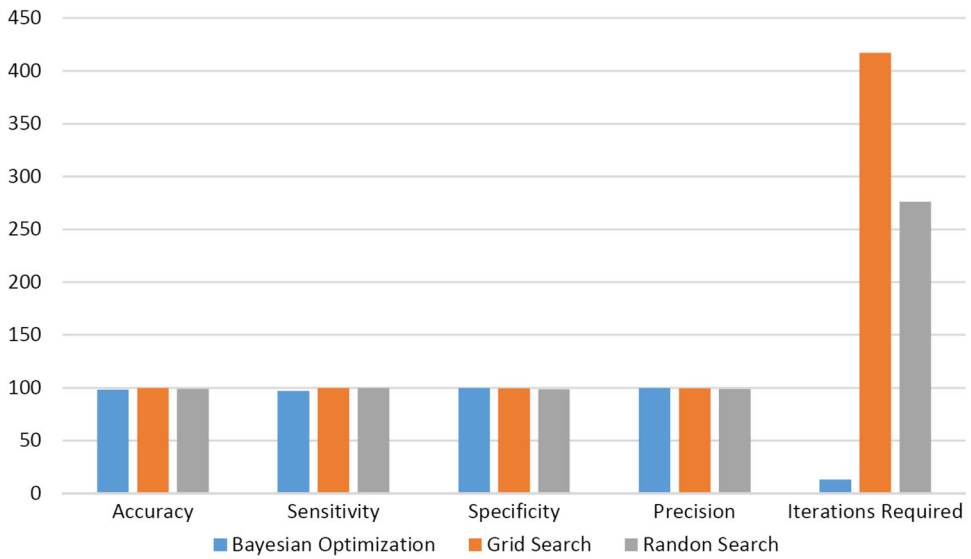

Fig. 10 Comparing Bayesian optimization with grid search and random search

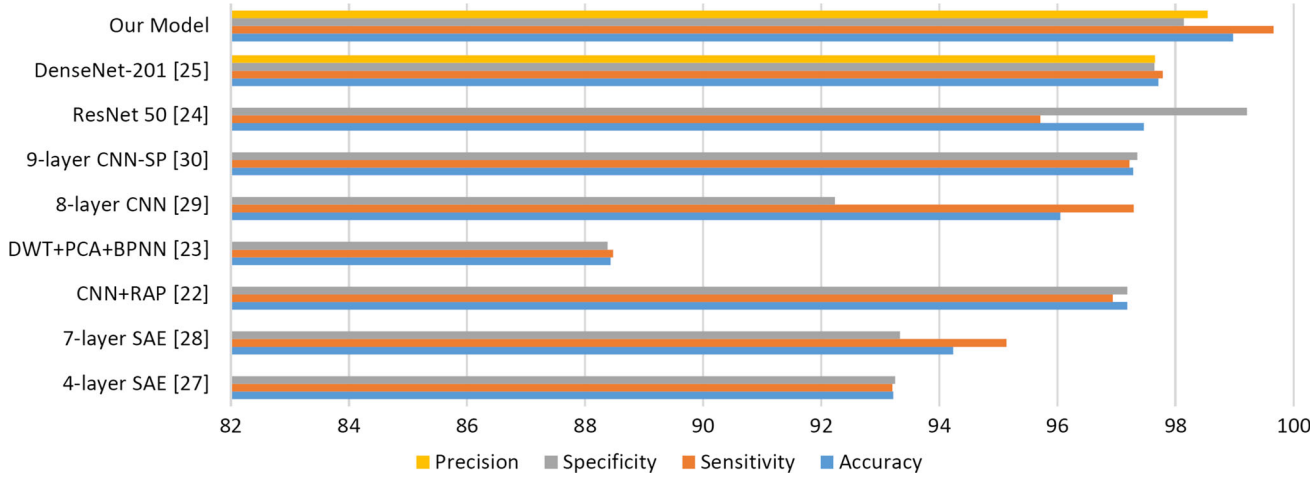

Fig. 11 Comparison with state-of-the-art methods

learn a highly complex hypothesis, our smaller but highly tuned model fits the problem better.

The nine-layer CNN-SP [30] outperforms the remaining methods with its balanced performance and manages to score above 97% in every metric. Stochastic pooling implemented here helps in regularizing the deep learning network. However, stochastic pooling also leads to a higher bias, hence under-fitting the data. Another approach that focuses on pooling methods is the CNN+RAP [22] method, which uses rank-based pooling to achieve higher translational invariance and scores about 97% in every metric.

Another approach, the eight-layer CNN [29], is very unbalanced but still scores no less than 92% in every metric. Subsequently, we find two SAE-based neural network approaches, with the seven-layer [28] being a bit more accurate but unbalanced than the four-layer [27]. Both methods still score above the 93% mark. Moreover, we observe the least accurate of them all, DWT+PCA+BPNN [23], with every metric being less than 89%.

Comparing to the state-of-the-art techniques, we find that our high performance is attributed to the reasoning-based hyper-parameter search we employ. Image augmentation also plays a crucial role in this success.

6 Conclusion

Early detection of CMBs attributes to early detection of related diseases and hence is of crucial importance. Our research focuses on a CNN-based approach, which is tuned using Bayesian optimization. We tested different methods to tune hyper-parameters and found our proposed method to be more efficient. Further comparing our model to the state-of-the-art methods, we again observed that our model scores better than the topmost performers, although being the smallest one (5 layers), especially when compared to ResNet-50 (50 layers) and DenseNet-201 (201 layers). This is based on the fact that we achieve an accuracy = 98.97%, sensitivity = 99.66%, specificity = 98.14%, and precision = 98.54%.

There are a few limitations to our model, which we plan to improve in the future. For instance, we can include more classes or separate classes for pathological brain diseases. Currently, we are using a sliding window to fragment images into smaller pieces, which are then fed to CNN. In future research, we can try to work directly with the 3D MRI images, bypassing the whole sliding window operation. This can bring more efficiency to our approach.

Furthermore, we will also focus on developing an unsupervised learning algorithm or semi-supervised learning [31] algorithm, as manual labeling of data is tedious if we wish to collect larger datasets for more sophisticated practical applications.

Acknowledgements This work was supported by Natural Science Foundation of China (61602250), Natural Science Foundation of Jiangsu Province (BK20150983), National Key Research and Development Plan (2017YFB1103202), Henan Key Research and Development Project (182102310629), Royal Society International Exchanges Cost Share Award UK (RP202G0230), Medical Research Council Confidence in Concept (MRC CIC) Award UK (MC_PC_17171), and Hope Foundation for Cancer Research UK (RM60G0680).

References

1. Yates, P., Sirisriro, R., Villemagne, V., Farquharson, S., Masters, C., Rowe, C., et al.: Cerebral microhemorrhage and brain β-amyloid in aging and Alzheimer disease. Neurology **77**(1), 48–54 (2011)
2. Fiehler, J.: Cerebral microbleeds: old leaks and new haemorrhages. Int. J. Stroke **1**(3), 122–130 (2006)
3. Nakata-Kudo, Y., Mizuno, T., Yamada, K., et al.: Microbleeds in Alzheimer disease are more related to cerebral amyloid angiopathy than cerebrovascular disease. Dement. Geriatr. Cogn. Disord. **22**(1), 8–14 (2006)

4. Martinez-Ramirez, S., Greenberg, S.M., Viswanathan, A.: Cerebral microbleeds: overview and implications in cognitive impairment. Alzheimer's Res. Ther. **6**, 33 (2014). https://doi.org/10.1186/alzrt263
5. Noorbakhsh-Sabet, N., Pulakanti, V.C., Zand, R.: Uncommon causes of cerebral microbleeds. J. Stroke Cerebrovasc. Dis. **26**, 2043–2049 (2017). https://doi.org/10.1016/j.jstrokecerebrovasdis.2017.07.012
6. Roberts, T.P., Mikulis, D.: Neuro MR: principles. J. Magn. Reson. Imaging **26**, 823–837 (2007). https://doi.org/10.1002/jmri.21029
7. Vernooij, M.W., Ikram, M.A., Wielopolski, P.A., Krestin, G.P., Breteler, M.M., van der Lugt, A.: Cerebral microbleeds: accelerated 3D T2*-weighted GRE MR imaging versus conventional 2D T2*-weighted GRE MR imaging for detection. Radiology **248**, 272–277 (2008). https://doi.org/10.1148/radiol.2481071158
8. Haacke, E.M., Xu, Y., Cheng, Y.C., Reichenbach, J.R.: Susceptibility weighted imaging (SWI). Magn. Reson. Med. **52**, 612–618 (2004). https://doi.org/10.1002/mrm.20198
9. Naka, H., Nomura, E., Wakabayashi, S., Kajikawa, H., Kohriyama, T., Mimori, Y., Nakamura, S., Matsumoto, M.: Frequency of asymptomatic microbleeds on T2*-weighted MR images of patients with recurrent stroke: association with combination of stroke subtypes and leukoaraiosis. AJNR Am. J. Neuroradiol. **25**, 714–719 (2004)
10. Tsushima, Y., Aoki, J., Endo, K.: Brain microhemorrhages detected on T2*-weighted gradient-echo MR images. AJNR Am. J. Neuroradiol. **24**, 88–96 (2003)
11. Lee, S.H., Bae, H.J., Kwon, S.J., Kim, H., Kim, Y.H., Yoon, B.W., Roh, J.K.: Cerebral microbleeds are regionally associated with intracerebral hemorrhage. Neurology **62**, 72–76 (2004). https://doi.org/10.1212/01.WNL.0000101463.50798.0D
12. Cordonnier, C., van der Flier, W.M., Sluimer, J.D., Leys, D., Barkhof, F., Scheltens, P.: Prevalence and severity of microbleeds in a memory clinic setting. Neurology **66**, 1356–1360 (2006). https://doi.org/10.1212/01.wnl.0000210535.20297.ae
13. Ateeq, T., Majeed, M.N., Anwar, S.M., et al.: Ensemble-classifiers-assisted detection of cerebral microbleeds in brain MRI. Comput. Electr. Eng. **69**, 768–781 (2018)
14. Cordonnier, C., Salman, R., Wardlaw, J.: Spontaneous brain microbleeds: systematic review, subgroup analyses and standards for study design and reporting. Brain **130**(8), 1988–2003 (2007)
15. Barnes, S.R.S., Haacke, E.M., Ayaz, M., Boikov, A.S., Kirsch, W., Kido, D.: Semiautomated detection of cerebral microbleeds in magnetic resonance images. Magn. Reson. Imaging **29**(6), 844–852 (2011)
16. Bian, W., Hess, C.P., Chang, S.M., Nelson, S.J., Lupo, J.M.: Computer-aided detection of radiation-induced cerebral microbleeds on susceptibility-weighted MR images. NeuroImage Clin. **2**, 282–290 (2013)
17. Fazlollahi, A., Meriaudeau, F., Villemagne, V.L.: Efficient machine learning framework for computer-aided detection of cerebral microbleeds using the Radon transform. Paper presented at, et al.: In: IEEE 11th International Symposium on Biomedical Imaging (ISBI); 2014. Beijing, China (2014)
18. Fazlollahi, A., Meriaudeau, F., Giancardo, L., et al.: Computer-aided detection of cerebral microbleeds in susceptibility-weighted imaging. Comput. Med. Imaging Graph. **46**(Part 3), 269–276 (2015)
19. Chen, H., Yu, L., Dou, Q., Shi, L., Mok, V.C., Heng, P.A.: Automatic detection of cerebral microbleeds via deep learning based 3D feature representation. Paper presented at: 2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI); New York, NY (2015)
20. Van den Heuvel, T.L.A., van der Eerden, A.W., Manniesing, R., et al.: Automated detection of cerebral microbleeds in patients with traumatic brain injury. NeuroImage Clin. **12**, 241–251 (2016)
21. Kaaouana, T., Bertrand, A., Ouamer, F., et al.: Improved cerebral microbleeds detection using theirmagnetic signature on T2*-phase-contrast: a comparison study in a clinical setting. NeuroImage Clin. **15**, 274–283 (2017)
22. Wang, Shuihua, Jiang, Yongyan, Hou, Xiaoxia, Cheng, Hong, Sidan, Du: Cerebral micro-bleed detection based on the convolution neural network with rank based average pooling. IEEE Access **5**, 16576–16583 (2017)
23. Hong, Jin, Zhihai, Lu: Cerebral microbleeds detection via discrete wavelet transform and back propagation neural network. Adv. Soc. Sci. Educ. Hum. Res. **196**, 228–232 (2019)
24. Liu, J., et al.: Detecting cerebral microbleeds with transfer learning. Mach. Vis. Appl. (2019). https://doi.org/10.1007/s00138-019-01029-5
25. Tang C., et al.: Cerebral micro-bleeding detection based on densely connected neural network. Front. Neurosci. 2019, **13**, Article ID: 422 (2019)
26. Kingma, Diederik P., Ba, Jimmy: Adam: A Method for Stochastic Optimization. In: 3rd International Conference for Learning Representations, San Diego, (2015). arXiv:1412.6980v9
27. Zhang, Y-D., Hou, X-X., Lv, YD., Chen, H., Zhang, Y., Wang, SH.: Sparse Autoencoder based deep neural network for voxelwise detection of cerebral microbleed. In: 22nd International Conference on Parallel and Distributed Systems: Wuhan, pp. 1229–1232. IEEE, China (2016)
28. Zhang, Y.-D., Zhang, Y., Hou, X.-X., Chen, H., Wang, S.H.: Seven-layer deep neural network based on sparse autoencoder for voxelwise detection of cerebral microbleed. Multimed. Tools Appl. **77**(9), 10521–10538 (2018)
29. Lu, Siyuan, Lu, Zhihai, Hou, Xiaoxia, Cheng, Hong, Wang, Shuihua: Detection of cerebral microbleeding based on deep convolutional neural network. In: 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP): Chengdu, pp. 93–96. IEEE, China (2017)
30. Chen, Y., et al.: Cerebral micro-bleeding identification based on nine-layer convolutional neural network with stochastic pooling. Concurr. Comput. Pract. Exp. (2019). https://doi.org/10.1002/cpe.5130
31. Al-Qurishi, M., Rahman, S.M.M., Alamri, A., et al.: SybilTrap: a graph-based semi-supervised Sybil defense scheme for online social networks. Concurr. Comput. Pract. Exp. **30**(5), e4276 (2018)

Piyush Doke is an undergraduate student at the Indian Institute of Technology Goa pursuing a B.Tech. in Computer Science and Engineering. He has a background in artificial intelligence and holds keen interest in deep learning techniques.

Dhiraj Shrivastava is an undergraduate student at Indian Institute of Technology (Banaras Hindu University) Varanasi and will be graduating in July 2020 with a B.Tech. in Mechanical Engineering. He has strong interest in the field of machine learning and computer vision. Dhiraj's research concentrates on the study and development of intelligent AI systems for healthcare.

Chichun Pan obtained master of management at Nanjing Normal University in 2001 and doctor of laws at Nanjing Normal Univer-

sity in 2012. He is currently working in Nanjing Normal University. His research interest is the commercial application of artificial intelligence, creativity and innovation.

Qinghua Zhou received his bachelors degree from the University of Sydney, Australia, in 2019. Currently, he is a Ph.D. student of the Department of Informatics, University of Leicester, UK. He is sponsored by the University of Leicester as a graduate teaching assistant (GTA). His research interests include machine learning and deep learning for medical data analysis.

Yu-Dong Zhang received his B.E. in Information Sciences in 2004 and M.Phil. in Communication and Information Engineering in 2007, from Nanjing University of Aeronautics and Astronautics. He received the Ph.D. degree in Signal and Information Processing from Southeast University in 2010. He worked as a postdoc from 2010 to 2012 with Columbia University, USA; and as an assistant research scientist from 2012 to 2013 with Research Foundation of Mental Hygiene (RFMH), USA. He served as a Full Professor from 2013 to 2017 with Nanjing Normal University, where he was the director and founder of Advanced Medical Image Processing Group in NJNU. Now he serves as Professor with Department of Informatics, University of Leicester, UK. His research interests include deep learning and medical image analysis.