IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 26, 2017, accepted December 31, 2017, date of publication January 25, 2018, date of current version March 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2796722

Image Classification Based on the Boost Convolutional Neural Network

SHIN-JYE LEE[1], TONGLIN CHEN[2], LUN YU[2], AND CHIN-HUI LAI[3]

[1]Institute of Technology Management, National Chiao Tung University, Hsinchu 30010, Taiwan
[2]National Pilot School of Software, Yunnan University, Kunming 650091, China
[3]Department of Information Management, Chung Yuan Christian University, Chungli 302023, Taiwan

Corresponding author: Chin-Hui Lai (chlai@cycu.edu.tw)

This work was supported in part by the Key Laboratory of Software Engineering of Yunnan Province under Grant 2017SE202, in part by the Applied Basic Research Foundation of Yunnan Province under Grant 2016FB104, in part by the Yunnan Provincial Innovation Team under Grant SJ2016-1, in part by the Natural Science Foundation of China under Grant 61402397, Grant 61663046, Grant 61379032, and Grant 61662085, and in part by the Ministry of Science and Technology of Taiwan under Grant MOST 106-2410-H-033-013.

ABSTRACT Convolutional neural networks (CNNs), which are composed of multiple processing layers to learn the representations of data with multiple abstract levels, are the most successful machine learning models in recent years. However, these models can have millions of parameters and many layers, which are difficult to train, and sometimes several days or weeks are required to tune the parameters. Within this paper, we present the usage of a trained deep convolutional neural network model to extract the features of the images, and then, used the AdaBoost algorithm to assemble the Softmax classifiers into recognizable images. This method resulted in a 3% increase of accuracy of the trained CNN models, and dramatically reduced the retraining time cost, and thus, it has good application prospects.

INDEX TERMS Convolutional neural network, ensemble learning, deep learning, boosting.

I. INTRODUCTION

The rise of big data and the rapid popularization of high-performance computing devices in recent years have contributed to the unprecedented development of machine learning. In particular, the breakthroughs of deep learning in the computer vision industry have made people discussing about artificial intelligence again. Regarding image identification, this study intends to artificially extract features, and convert the features of an original image into useful features characterized by lower dimension and less noise. The Random Forest [1], Support Vector Machine (SVM) [2], and other supervised learning algorithms are adopted to solve classification problems or adopted to train classifiers; however, there are limitations in the artificially designed feature extraction methods: meaning they are poor in generalization and have high labor costs.

As a bio-inspired neural network, the Convolutional Neural Network (CNNs) [3] has become the most successful deep learning model in the computer vision industry; however, CNNs tend to cause overfitting problem with poor generalized capability due to its complicated mechanism. As an iterative integration method, Adaboost [4] finds weak classifiers through iteration, and integrates them into strong classifiers. In addition to being efficient in generalization, Adaboost can effectively control the quantity of weak classifiers, which greatly increases execution efficiency through integration; therefore, it has also been applied to real-time object testing. Hence, this study adopts Adaboost for the iterative training of several convolution functions and integrates them into a Boost Convolutional Neural Network. This effectively reduces parameter redundancy, which results in greater generalization and higher training efficiency.

In a narrow sense, a neural network comprises multiple hidden layers can be identified as a kind of deep learning; in a broad sense, it is a connectionist learning model equipped with layered feature extraction. Starting with the original features of data, deep learning minimizes the loss function by extracting features layer by layer. Through multilayered feature conversion, it gradually transforms the original data features, such as the pixel features in images, into abstract marginal and formal features, in order to enhance the performance of the network. In recent years, deep learning has been quickly promoted in the machine learning industry, and remarkable theoretical and practical achievements have been continually attained. Specifically, outstanding breakthroughs have been made in various areas, such

as computer vision, natural language processing, and sound recognition.

The development of deep learning can be divided into three stages. From the 1940s to the 1960s, deep learning was called the control theory. While the theoretical development of biological learning and the establishment of the perceptron triggered the first wave of interest in artificial neural networks, as the perceptron could not be used to learn linearly inseparable data, it was resisted by many researchers, which resulted in the low ebb of the first stage. The second wave started with connectionism in 1980, and ended in 1995. In 1974, Paul Werbos proposed the back propagation algorithm, and solved the problem of linear inseparability in the neural network model. However, as the quantity of training data was small and the performance of the computers was poor at that time, there was a serious problem in most of the deep learning models - overfitting. At the same time, great progress was made in other machine learning fields. For instance, kernel functions, graphic models, and SVM brought positive results in many important tasks. Consequently, deep learning was neglected by most people, which marked the low ebb of the second stage. In 2006, Hinton and Salakhutdinov [5] from the University of Toronto displayed a neural network model called the autoencoder in the magazine Science. By adopting an unsupervised learning algorithm to initialize networks layer by layer, they efficiently improved the generalization of the deep network. Since then, many researchers have used the same method to revitalize a large number of the deep learning models, which were challenged in the 1980s. This marked the entry into the third wave of deep learning.

Based on the visual system of mammals, the convolution neural network (CNN) is a neural network used exclusively to process data with a lattice structure. In 1959, Hubel and Wiesel discovered the visual cortex cells of mammals, and proposed the concept of a partial receptive field. Inspired by this, Fukushima and Kunihiko [6] put forward Neocognitron in 1984, which can be regarded as the prototype of the modern convolution network. In the 1990s, Le Cun *et al.* published relevant papers to define the modern structure of CNN, and improved it afterwards. Regarding handwritten number classification, LeNet-5 [7], the first convolution network that could be applied in reality, was created. With little preprocessing, the network can directly learn the number image classifications in the original pixels. However, the training data were inadequate, and the computation ability of the computer was weak; thus, LeNet-5 was not effective in handling complicated problems. In 2009, Lee *et al.* [8] integrated the convolution neural network with the deep belief network to form the Convolutional Deep Belief Network (CDBN). In 2012, Krizhevsky *et al.* [3] adopted AlexNet to bring the error rate of image classification down from 26.3% to 15.2% in ILSVRC-2012. In 2014, a Google team used GoogLeNet to achieve a lower error rate (6.67%) [9] in ILSVRC-2014. In 2015, the convolution network of Microsoft MSRA brought the error rate (top-5) of the ImageNet 2012 data concentration down to 4.94%, thus, surpassing humans in terms of recognition [10].

The first step in this paper is to introduce the basic knowledge of the convolution, pooling, weight decay, and dropout of the convolution neural network, as well as the theories and knowledge of ensemble learning. With the convolution neural network and the learning principles of Adaboost, this paper takes the initiative to propose the Boost Convolutional Neural Network, and conducts experimentation and analysis of image classification with the benchmark datasets of CIFAR-10 image classification. This paper consists of six chapters, and its overall structure is, as follows. Chapter 1 elaborates on the research background and significance, and gives detailed descriptions of the deep learning abilities and current developments of the convolution neural network. Chapter 2 elucidates cross entropy, convolution, pooling, weight decay, dropout, and other basic knowledge of the convolution neural network. Chapter 3 focuses on the theory of ensemble learning, the principle of Boosting, and Adaboost. Chapter 4 provides a detailed introduction to the network model structure and training methods of BoostCNN. Through comparative experiments of the CIFAR-10 image classification datasets, Chapter 5 analyzes the advantages and disadvantages of BoostCNN. Chapter 6 offers the conclusion of this paper, and presents prospects for future studies of the same topic.

II. CONVOLUTIONAL NEURAL NETWORKS

The Artificial Neural Network (ANN), also called a neural network, is a computational model based on the biological neural network. Its core concept originated in the connectionism of cognitive science: a huge number of simple computational units are connected to show intelligent behaviors. Such a concept is applicable to the neurons of the biological neural system and the computational units of computational models. Therefore, exploration into the neural system is inspired by the brains of organisms to develop an intelligent system, and then, delve into the intelligent behaviors of the organisms in the intelligent system.

The neural cells in a brain can be simplified into two states - fire and non-fire (restriction). The intensity of signal transmission remains unchanged, and only the frequency changes. Neural cells use certain methods to aggregate all the signals added into neurons, and if the total number of the signals exceeds a certain threshold, then it will stimulate the neural cells and trigger the state of fire. In this case, electrical signals will be sent to other neural cells, and if the total number is lower than the threshold, the neural cells will remain restricted, without sending any signals. Figure 1 shows a standard neuron model, which is comprised of two parts. The first part is the accumulation of signals, where the input signals (input data) are accumulated for a sum. As shown in (1), each weight (w_i) matches a data dimension (x_i), while (w_0) as a bias is equivalent to the intercept term or constant term of the function. Whilst the constant is set to "1" as the input of 0^{th} dimension, the bias is processed as the weight of 0^{th}

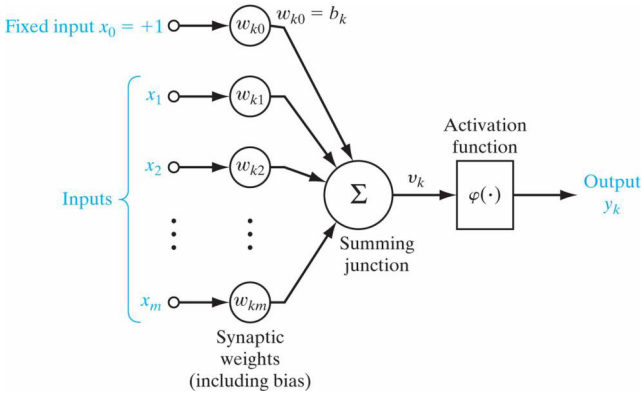

FIGURE 1. Standard neuron model.

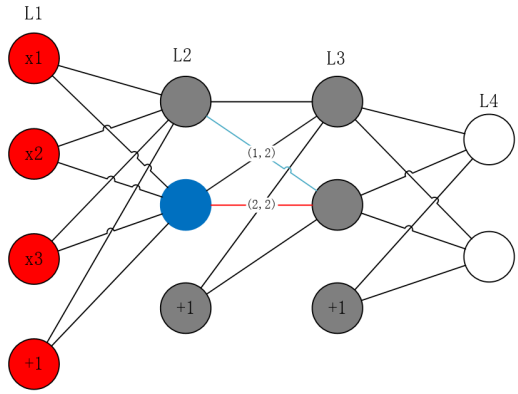

FIGURE 2. Feedforward neural network.

dimension. This is also called affine transformation.

$$Z = bias + \sum_{i=1}^{m} x_i w_i = \sum_{i=0}^{m} x_i w_i \tag{1}$$

The second part is the activation of the function, where the obtained activation value is used for the non-linear compressed transformation to extract a non-linear eigenvalue. The frequently-used activation functions include ReLU, Sigmoid, and Tanh.

A neural network is a network based on the interconnection of artificial neurons.

A. FEEDFORWARD NEURAL NETWORKS

The feedforward neural network (FNN) or multilayer perceptron (MLP) is a neural network that allows the feedforward connection of neurons. The input of data is called the input layer, while the output of results is called the output layer; the layers between the input layer and the output layer are called hidden layers. If the number of hidden layers is large, then the network will also be called a deep neural network.

1) FORWARD PROPAGATION

In the test or execution stage of the feedforward neural network, the data X are calculated through function $f(x)$ in the middle layer, which are finally output as y on the output layer. In this process, the information cannot be transmitted backward or horizontally. Figure 2 shows a four-layered neural network, where the first layer (L1) is the input layer; L4 is the output layer; L2 and L3 are hidden layers; $a_{i,j}^{(l)}$ refers to the connection weight of "i" (ordinal number) neuron on Layer I and "j" (ordinal number) neuron on Layer I + 1; a_j^l indicates the connection between the bias on Layer I and "j" neuron on Layer I + 1; and a_i^l implies the activation value (output value) of the "i" neuron on Layer I, and the activation value of the blue neuron in the picture is $a_2^{(2)}$.

To facilitate the description, if I = 1, then the input feature of the "i" dimension on the input layer will also be symbolized as "a", which means $a_i^{(1)} = x_i$. As shown in (2), it is the activation expression of "j" neuron on Layer I + 1. In the process, $f(x)$ is the activation function:

$$a_j^{(l+1)} = f(\sum_{i=1}^{m} a_i^{(l)} w_{i,j}^{(l)} + b_j^{(l)}) \tag{2}$$

The activation value of the blue neuron in the picture can be obtained with (3) :

$$a_2^{(2)} = f(x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_3 w_{3,2}^{(1)} + b_2^{(1)}) \tag{3}$$

2) BACKWARD PROPAGATION

The back-propagation algorithm [11] is called the BP algorithm for short, which was separately created by several individuals in the 1970s to the 1980s. Its initial form was the linear algorithm, as developed by Bryson and Ho in 1969. Rumelhart, Williams, and Hinton created the non-linear BP algorithm in 1981; however, as their first test was so disappointing, they decided to stop using it. In 1986, they cited a persuasive example to demonstrate that the method was useful and highly promising in the learning of the multi-layered non-linear neural network, and in the late 1990s, the most rigorous researchers began to stop adopting the BP algorithm, and the appearance of the Support Vector Machine (SVM) accelerated the overlook of the BP algorithm. As a psychological model, it was still widely applied by psychologists. The main reasons why the BP algorithm lost popularity were the slow operation of a computer and the inadequate number of training data. Worse still, the deep network model at that time was too small, there were not any rational strategies for weight initialization, and gradient vanishing was neglected. As a result, the BP algorithm was ineffective in the deep network, which hindered its development. Today, with the immensely improved performance of computers, the problem of datasets has been solved; therefore, the BP algorithm has become a mainstream training method of deep learning.

The core concept of the BP algorithm is to use the chained derivation rule to obtain the backward corresponding gradients of the parameters of the neural network layer by layer. Network training is divided into two steps: the first step adopts uniform distribution or Gaussian distribution of the random initialization of weight parameters of the neural network; the second step uses the training data for iterative training to obtain the loss value of the neural network model.

The loss value is then used for the chained derivation, and the error gradient is transmitted layer by layer from the last layer backward to the input layer. Then, the weight values of all layers are updated according to the gradient algorithm until the requirements are met or it exceeds the maximum number of iterations.

For each output unit "i" on the "n" (ordinal number) layer, the residual error is calculated with (4):

$$da_{\mathrm{i}}^{n} = J'(x; w) \tag{4}$$

The residual errors from the second layer to the "$n-1$" layer and one at the "i" node on Layer I are shown in (5):

$$da_{\mathrm{i}}^{l} = (\sum_{j=1}^{m} da_{j}^{(l+1)} w_{i,j}^{(l)}) f'(\sum_{i=0}^{m} a_{i}^{(l-1)} w_{i,j}^{(l-1)}) \tag{5}$$

The partial derivatives from the "i" neuron on Layer I and the "j" neuron on Layer I + 1 are shown in (6). The partial derivative gradient of the connection between "i" neuron on Layer I and "j" neuron on Layer I + 1 is shown in (7).

$$\frac{\partial J(w)}{\partial w_{i,j}^{l}} = da_{j}^{l+1} f'(\sum_{i=0}^{m} a_{i}^{(l)} w_{i,j}^{(l)}) a_{i}^{l} \tag{6}$$

$$\frac{\partial J(w)}{\partial b_{j}^{l}} = da_{j}^{l+1} f'(\sum_{i=0}^{m} a_{i}^{(l)} w_{i,j}^{(l)}) \tag{7}$$

B. CONVOLUTION OPEARTION

The convolution network is one of the most successful biology-inspired artificial intelligence networks. Although the convolution network has been directed by many other fields, some key design principles of the neural network are from neural science. To determine how the visual system of mammals worked, neurophysiologists David Hubel and Torsten Wiesel conducted a cooperative study, and found that specific neurons in the visual system of cats were only sensitive to some forms, and that these neurons could still be activated even if these forms were removed. After years of exploration, they finally figured out the operation principles of the primary visual cortex (V1) of the visual system of mammals, and their finding earned them the Nobel Prize.

The visual system of mammals is highly similar to a convolution operation. Suppose that two-dimensional image "I" is taken as the input, and the two-dimensional convolution kernel is represented by "K"; the convolution of the input image is, as follows:

$$\mathrm{S}(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-\mathrm{n}) \tag{8}$$

Usually, Equation (8) is easily implemented in a machine learning library. In the above equation, convolution must reverse the convolution kernel, and then, aggregate the weights. In other words, the number of input indices will increase, while that of the convolution kernel indices will decline. The convolution kernel is reversed to meet the law of commutation of convolution; however, the law of commutation is less performed in a neural network, thus, it is

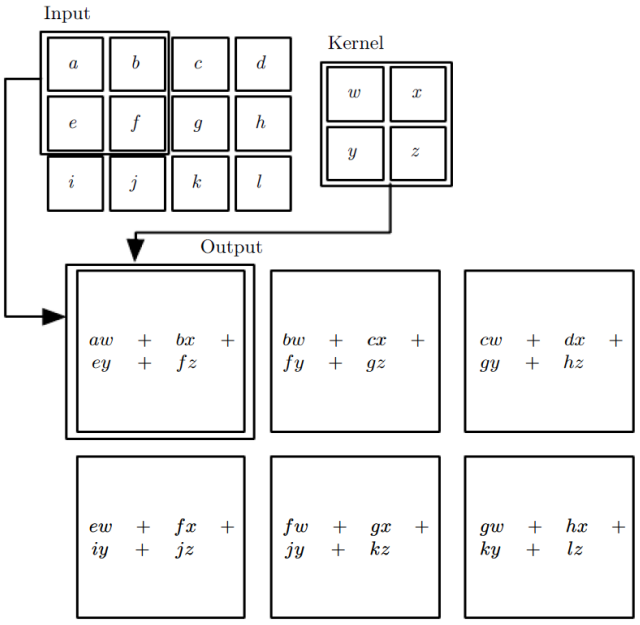

FIGURE 3. Image convolution.

cross-correlation that is achieved in a neural network, as shown in (9). While cross-correlation is similar to convolution, it does not require reverse convolution.

$$\mathrm{S}(i,j) = (\mathrm{I} * K)(i,j) = \sum_{m}\sum_{n} I(i+m, j+n)K(m,n) \tag{9}$$

Despite cross-correlation being achieved, it is still called convolution in most machine learning libraries. Figure 3 shows a valid convolution operation without the reversion of the convolution kernel. From the perspective of machine learning, convolution has two important concepts – sparse connectivity and parameter sharing

1) SPARSE CONNECTIVITY

As shown in Figure 4, in a traditional neural network, each neuron is connected to all neurons on the upper layer, and such architecture is called full connectivity. As revealed in Figure 4, a neuron is merely connected with some neurons on the upper layer, and such architecture is called sparse connectivity [12]. The most distinctive difference between the two connections lies in the great difference in the quantity of the parameters. Parameter quantity is the most direct reflection of the efficiency of a model in machine learning, and the efficiency of the model has direct effect on overfitting; hence, sparse connectivity is an effective way to prevent overfitting. For example, in image processing, an input image may have millions of pixels, and there may not seem to be any connection between the individual pixels and the images. However, we can further process images by detecting the small, but significant, corner features in hundreds of pixels. The reduction of parameters also indicates the reduction of the necessary memory resources and computational operations, which will greatly increase efficiency.

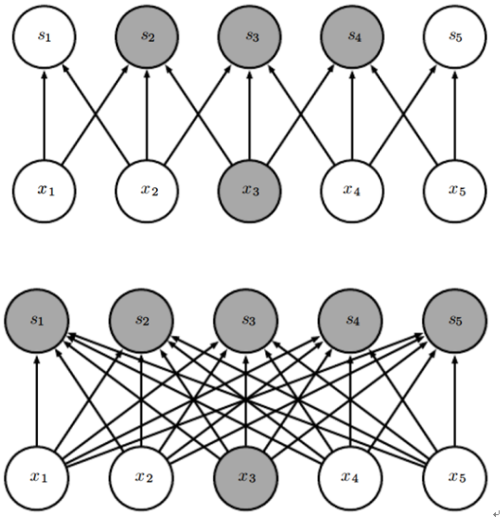

FIGURE 4. Sparse connectivity and full connectivity.

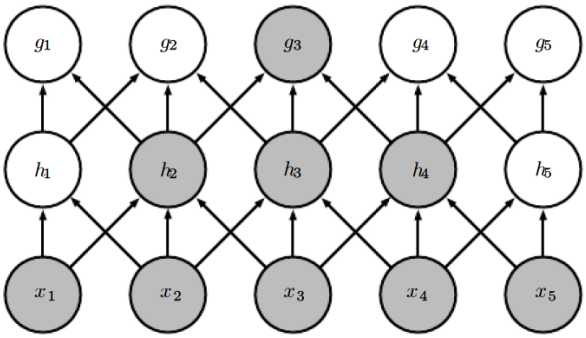

FIGURE 5. Multi-layered sparse connectivity network.

Suppose that there are "m" input units and "n" output units, and these $m \times n$ parameters must be connected for one transmission. In practice, each calculation requires O($m \times n$) temporal complexity. If each output unit is confined to the sparse connectivity of the "k" connectivity parameters, then the number of necessary parameters will be reduced to $k \times n$ and temporal complexity will be O($k \times n$). Usually, "k" is much smaller than "m", thus, the increased efficiency is very appreciable.

On a singular layer, the information of partial connection is incomplete; however, in the deep convolution network, the sparse network complements the input information of the lower-layer network through an indirect connection. As shown in Figure 5, while each neuron is partially connected, g3 still completes all the input information through h2, h3, and h4. Due to the multi-layered sparse connectivity, complicated network interactions can be efficiently expressed through the partial sparse connectivity alone.

2) PARAMETER SHARING

Parameter sharing means that the same parameter is used in several parts of a model. In the neural network, there is a

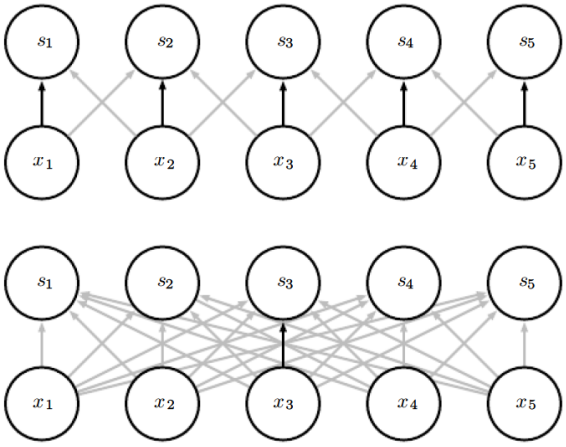

FIGURE 6. Parameter sharing.

synonym for parameter sharing - tied weight. This means that the weight used in one part will be tied to other parts in a neural network. In the convolution network, the convolution kernel conducts convolution in all parts of the image; therefore, convolution does not learn the features in a specific location, but extracts features from all areas of data. For instance, a set of convolution parameters can extract vertical edge features, and these features may frequently appear in all parts of the image; hence, the convolution scan is in fact a filtering test on the vertical edge features of the image. As shown in Figure 6, the black arrow refers to a specific connectivity weight. In the convolution network in Figure 6, while the black arrow is used in each input neuron and corresponding output neuron, in the full connectivity network in Figure 6, the black arrow is merely used in the connection between Neuron x3 and Neuron s3.

Though parameter sharing affects the calculation time of the neuron network, the temporal complexity is still O($k \times n$). However, it does significantly reduce the number of parameters to be stored. Originally, $k \times n$ connectivity weights had to be stored, but only "k" parameters have to be stored now. Taking the example of the recognition of an image of $1000 * 1000$ pixels: suppose that there are 10,000 neurons on the first hidden layer; in the full connectivity network, 10 billion parameters will be needed on this layer alone. Suppose that the convolution network is adopted; the convolution kernel will consist of 100 parameters ($10 * 10$); if sparse connectivity is used, then the number of parameters will be reduced to 1 million. Suppose that parameter sharing is adopted; only 100 parameters must be stored. While the reduction from 10 billion to 100 is remarkable, in reality, several convolution kernels would be used to extract various features. Suppose that 100 convolution kernels are used; the number of parameters to be stored will be only 10,000.

C. POOLING OPERATION

In a typical convolution network, a complete layer of a convolution network comprises three stages. The first stage is

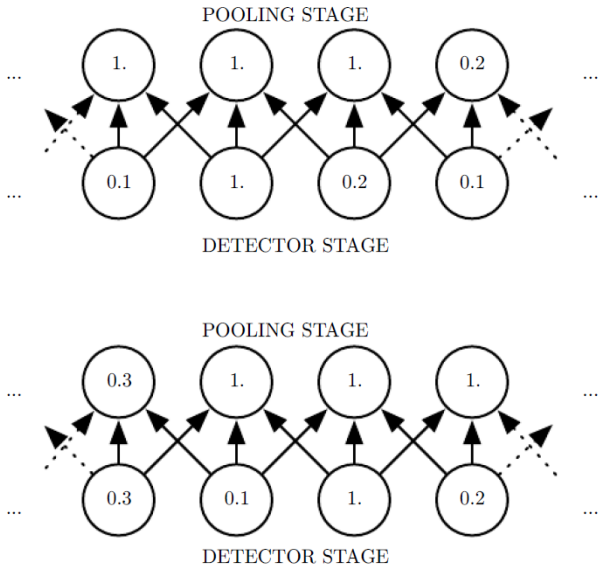

FIGURE 7. Maximum pooling.

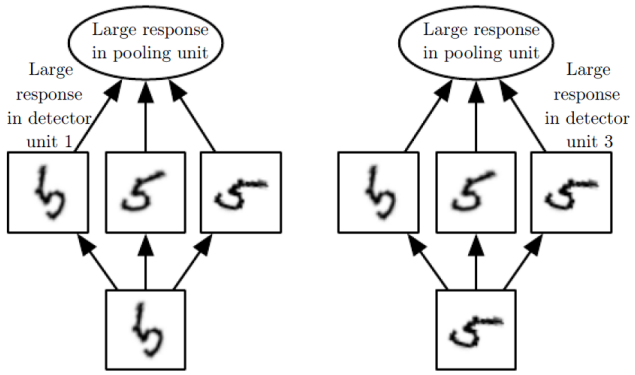

FIGURE 8. Rotational invariance brought by maximum pooling.

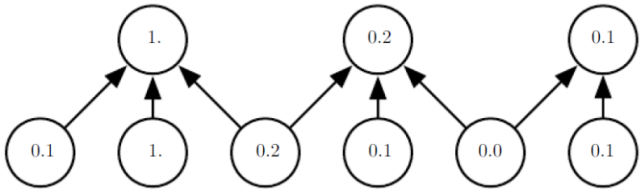

FIGURE 9. Application of maximum pooling to downsampling.

the convolution layer, where the convolution operation is conducted to form a group of feature images. The second stage is the detection layer, where each feature value is sent to a non-linear activation unit to be activated. The third stage is the pooling layer, where the features extracted on the lower layer are selected for sampling, which makes the network smaller. The pooling operation is very easy; for example, max pooling [13] outputs the maximum in the sampling area after convolution mapping; average pooling [14] outputs the averages in the sampling area after convolution mapping. What is worthy of attention, is that pooling not only makes the network smaller, it also obtains the invariant features of the input data. When the existence of some features is considered, rather than the locations of these features, then partial invariance will be a highly useful property. For example, to determine whether there is a human face in an image, the exact locations of the eyes are not required, only that there is an eye on the left side of the human face and the other is on the right side. As the blurry location strengthens the network's ability to resist noise, the generalization of the model improves. In a neural network, we hope to add future knowledge, such as invariance, to quicken the training of the network and enhance network performance.

1) TRANSLATION INVARIANCE

As shown in Figure 7, the maximum pooling operation is adopted to output the maximum of the three neighboring units on the detection layer onto the pooling layer. When the input values are translated, as shown in Figure 7, the output on the pooling layer will remain nearly unchanged. As the maximum pooling operation is only sensitive to the maximum eigenvalue, it does not consider accurate location, and such insensitivity equips the network with translation invariance.

2) ROTATIONAL INVARIANCE

According to the abovementioned, applying pooling to the same convolution detection layer equips the model with translation invariance. If pooling is applied to different convolution detection layers, and the maximum pooling of different convolution results is conducted, then rotational invariance will be achieved. As shown in Figure 8, suppose that there are three convolution detectors, and each detector can extract the number image "5" of different directions. If the number "5" appears in the input, the corresponding neuron will be activated. As shown in the left image of Figure 8, a "5" of upward rotation is input. After convolution testing, the first convolution unit will be activated. As shown in the right image, when a "5" of downward rotation is input, the third convolution unit will be activated. However, as long as a convolution unit is activated, the number "5" can be identified after maximum pooling.

The abovementioned pooling operation is like an alternative convolution operation. After the convolution layer, every two units are pooled once from the upper left side to the lower right side; however, this method has high calculation costs. As only the maximums or averages of neighboring nodes are required, such intensive pooling is unnecessary. Is it possible to achieve jumping pooling? For instance, a maximum is selected from Units 1-3, and then, one from Units 4-6. This means that the pooling operation is conducted among three units. Such a method is called downsampling. As shown in Figure 9, the width of pooling is set as "3", and two units are jumped over for sampling. This method effectively increases the efficiency of calculation. Suppose that "k" is a jumping unit; about "k" times of input neurons will then

be reduced on the lower layer. Lowering the lower dimension can effectively reduce the number of connectivity parameters on the lower layer, which reduces memory consumption and increases learning efficiency.

Pooling can also be taken as a basic approach to process extended input. In image identification, the quantity of image data is different; however, in a neural network, the network is stationary. Suppose that the input unit is 100; then only 100-pixel images can be input; however, if there are 110-pixel images, these images must be shrunk or cut; if there are 90-pixel images, these images must be magnified or complemented; to avoid directly cutting images, pooling is adopted to set a fixed size for input data.

D. BATCH NORMALIZATION

Batch Normalization [15] has become one of the most useful techniques to optimize a deep neural network. Highly simple and available, it can be combined with other algorithms to greatly speed up training of the deep model. Suppose that "m" (cardinal number) data are adopted for training: $H_{i,j}^{(k)}$ refers to the output value of the "i" neuron on "j" layer in the training of "k" (ordinal number); $\mu_{i,j}$ represents the average output value of the "i" neuron of the "j" layer among the data; $\sigma_{i,j}$ indicates the standard deviation of the output of the "i" neuron on the "j" layer among the data. The output value after batch normalization is shown in (10):

$$H'^{(k)}_{i,j} = \frac{H^{(k)}_{i,j} - \mu_{i,j}}{\sigma_{i,j}} \tag{10}$$

The average output value of neuron ($\mu_{i,j}$) is shown in (11):

$$\mu_{i,j} = \frac{1}{m}\sum_{k=1}^{m} H^{(k)}_{i,j} \tag{11}$$

The standard deviation of the output value of neuron ($\sigma_{i,j}$) is shown in (12):

$$\sigma_{i,j} = \sqrt{\delta + \frac{1}{m}\sum_{k=1}^{m}(H^{(k)}_{i,j} - \mu_{i,j})^2} \tag{12}$$

In the equation, δ is a small constant that aims to prevent the generation of $\sqrt{0}$. The purpose of batch normalization is simple – adjusting the input data on each layer of the neural network into a standard normal distribution with an average value of "0" and a variance of "1". Suppose that sigmoid is taken as the activation function of a neuron; if the output value is high, then its function will enter the saturation area, and the derivative will become nearly "0". Even if we know that the neuron needs to be corrected, training will remain impossible due to the small gradient. As shown in Figure 10, the value of the sigmoid activation function in $[-2, 2]$ is a similarly linear area. The BN algorithm strives to normalize the input value, as much as possible, in the narrow area of the activation function.

The BN algorithm is adopted to project the input value into the linear area of the activation function. If the input values of all layers are merely normalized into a similarly linear area, then the performance of the deep network will drop greatly. Therefore, the BN algorithm includes another step – magnifying the normalized data and translating them into the non-linear area. As shown in (12), two learnable parameters (γ and β) are adopted to adjust the output value of normalization.

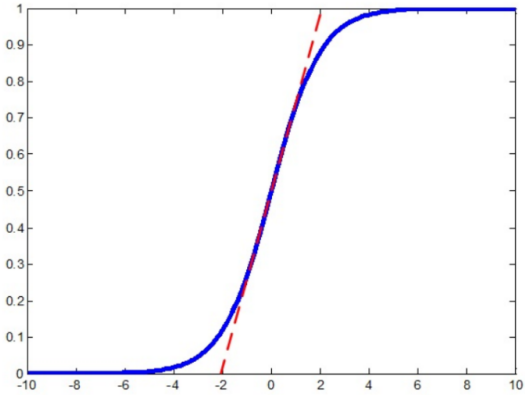

FIGURE 10. Comparison between sigmoid activation function and linear function.

$$X^{(k)}_{i,j} = \gamma_{i,j} H'^{(k)}_{i,j} + \beta_{i,j} \tag{13}$$

In this equation, variables "γ" and "β" are the learning parameters, which allow new variables to have any average value and standard deviation. In addition to reflecting the non-linear performance of old parameters, new parameters can eliminate the connection between layers, and thus, have a relatively independent learning approach. In the old parameters, the average value and variance of "H" depended on the complex connection among the parameters of the layer lower than "H". In the new parameters, the average value and variance of $(\gamma \cdot H' + \beta)$ are merely determined by the layer. New parameters can be easily learned through gradient reduction.

As only the data of batch sampling are normalized in training, the average and variance of these data are not those of all data. Therefore, the average value and variance of each batch of training data must be cumulated after normalization. After training, the average value and variance of all data are obtained, and used in testing.

Running Averages: As it is too troublesome to cumulate the average value and variance of each batch of data, in practice, the average value and variance in running are often used to replace that of all data. As shown in (14) and (15), the attenuation factor is adopted for the attenuation accumulation of average value and variance, in order to prevent counting the average value and variance of all batches.

$$\mu = decay \cdot \mu + (1 - decay)\mu_{sample} \tag{14}$$

$$\sigma = decay \cdot \sigma + (1 - decay)\sigma_{sample} \tag{15}$$

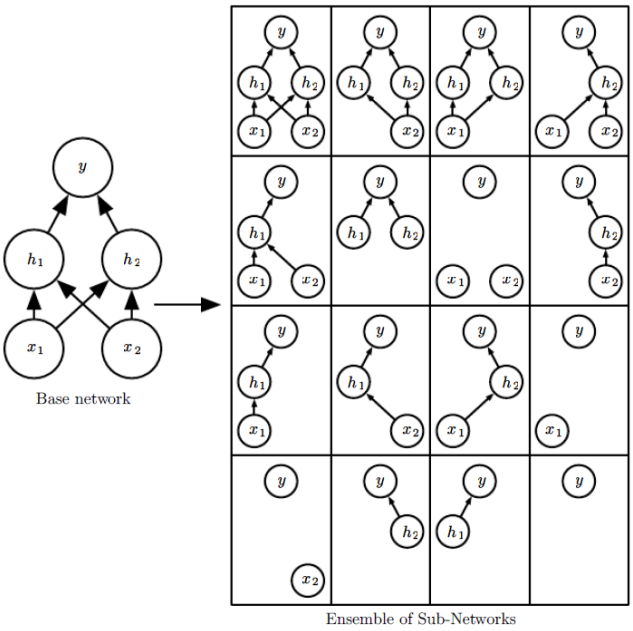

FIGURE 11. Structural decomposition of example neural network.

E. DROPOUT REGULARIZATION

Dropout [16] is a regularization model with low calculation cost and strong deep learning ability, and its concept is mainly derived from the Bagging (short for bootstrap aggregating) [17] of ensemble learning. As shown in Figure 11, the neural network in the left picture can be divided into 16 different sub-networks, and Dropout will integrate the weak sub-neural networks into the neural network.

According to Bagging, different learners are required. Different training sets are sampled from the training data, and each learner uses different training datasets to obtain learners with different ''views''. In Dropout, a binomial distribution ''neuron activation'' probability is set for each neuron in training; if the value is ''0'', then the current neuron is restricted; if the value is ''1'', then the current neuron is available. As shown in Figure 12, when the activation probability is 0.5, the quantity of the neurons and parameters (quantity of connectivity weights) of the weak neural network, as constructed by Dropout, is greatly restricted.

Dropout is divided into two stages – training and testing (execution). In the training stage, in comparison with the traditional feedforward neural network, it only has an extra neuron sampling. At the same time, a random gradient reduction or minimum gradient reduction is adopted to randomly activate a certain number of neurons. One data or one batch of data are then used for the backward error deviation of the activated neurons to correct the weights. Then, the neurons are randomly activated to train the network. In the stage of testing or execution, the neuron sampling is removed, and the traditional neural network is employed for testing.

Dropout can be seen as a form of Bagging in deep learning. Different network structures are constructed through neuron sampling, and different data are taken as samples to train different weak neural networks; however, there is a highly different part in this process. After Dropout training, each hidden unit in the neural network must learn to cooperate with different sampled neurons, which renders the neurons more robust and drives them to obtain useful features, rather than relying on other neurons to correct their errors. In Dropout, a hyper-parameter of neuron sampling probability (p) is adopted. While the default value is set as ''0.5'', it is not a norm, and thus, it must be repeatedly tested with different data and networks.

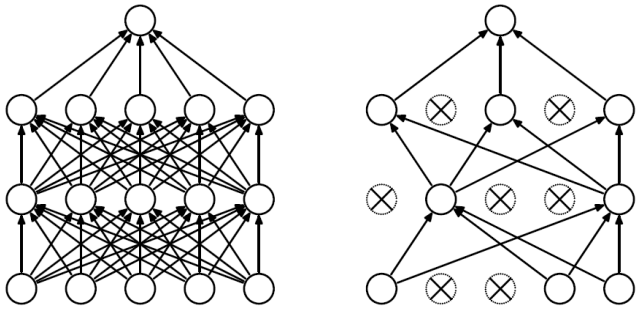

FIGURE 12. Dropout neural network model.

F. SUMMARY

The first step in this chapter is to introduce the work principles of neurons. In Section 2.1, the neurons are connected to form the neural network, and its forward and backward transmissions are the general work mode of all neural networks. Section 2.2 elucidates the biological inspiration of convolution, as well as the application of convolution in machine learning, parameter sharing, and coefficient connectivity. Section 2.3 introduces the pooling operation, and gives a detailed description of how to learn translation invariance and rotatory invariance through pooling. In Section 2.4, focus is placed on how batch normalization dynamically decouples all layers of the neural network and facilitates network training. Section 2.5 shows how to use Dropout to prevent overfitting of the neural network, and illustrates the significance of the machine learning and biological significance of Dropout.

III. BOOSTING ENSEMBLE LEARNING

As shown in Figure 13, ensemble learning [18] completes learning tasks by constructing and cooperating with several learners, meaning the so-called ''collectivism'', where a group of individual learners are trained, and a strategy is then adopted to combine them. Individual learners can be if the same type or of different types; if they are of the same type, then integration will be called homogeneous, and the individual learners in homogeneous integration are called base learners. For instance, a neural network is a typical network established through the integration of neurons. If the individual learners are different, then the integration will be called heterogeneous, and the individual learners of heterogeneous integration are generated through different learning

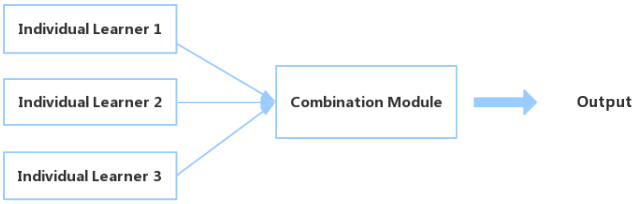

FIGURE 13. Structure of the ensemble learner.

algorithms. In this case, the individual learners will not be called base learners, but component learners. Homogeneous integration is the same as the neural network in the final structure, and the reason why the neural network is separated from ensemble learning lies in the difference in its organization. Ensemble learning is a ''bottom-to-top'' construction: the first step is to obtain learners; then, attention is paid to how to organize them to form an integrity. Conversely, the neural network is a ''top-to-bottom'' construction: the first step is to integrate the learners; then, all the learners in the learning structure are adjusted.

Ensemble learning combines several learners, and often shows better generalization than individual learners, especially for weak learners. Hence, many theoretical studies of ensemble learning have been conducted for weak learners. Sometimes, base learners are directly called weak learners. In addition, ensemble learning may bring three strengths, which are described, as follows, and shown in Figure 14.

First, ensemble learning can statistically offset the probability of the wrong selection of the hypothesis space of individual learners. As the hypothesis space of learning tasks is usually large, and several hypotheses may share the same performance in the training set, wrong individual learners may result in poor generalization. Therefore, the integration of several learners will reduce such risk.

Second, in terms of computation, ensemble learning can reduce the risk of overfitting. As an individual learning algorithm tends to be trapped by a partial optimum, the generalization of some partial optimums may be poor, thus, integration based on several operations can reduce the risk of poor partial optimums.

Third, regarding expression, ensemble learning can expand the hypothesis space of individual learners, and thus, facilitate the learning of an approximating function. The true hypothesis of some learning tasks may not be in the hypothesis space considered by the current learning algorithm; in this case, individual learners will be invalid; however, if several learners are combined, it is possible to acquire a better approximating function, as the corresponding hypothesis space has been expanded.

A. BOOSTING ALGORITHM

Boosting [19] is an algorithm concept that combines weak classifiers through certain methods to form a strong classifier with high classification performance. In 1990, Schapire used the PAC theory to demonstrate that, provided that the

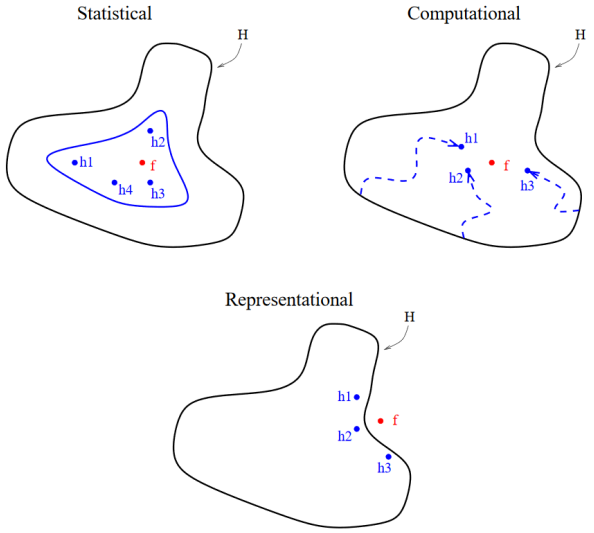

FIGURE 14. Three fundamental reasons why an ensemble may work better than a single classifier [18].

classification performance of a weak classifier was slightly higher than 0.5, it would be possible to combine several weak classifiers to form a strong classifier; moreover, he developed the first demonstrable multi-item Boosting learning algorithm.

The work mechanism of the Boosting algorithm is, as follows: the first step is to train the first learner in the initial training data; however, this learner is only effective in identifying some data; then, the distribution of the training data is adjusted according to the previous learner, and a higher sampling probability is given to data that are difficult to identify. With a focus on these data in the next round of learning; the above steps are repeated until the number of learners reaches the preset number; finally, the weights of all the trained learners are aggregated. The process of Boosting classification is, as follows:

1) Train the first weak classifier (h1) in the initialized training dataset;

2) Combine the poorly-trained data of the weak classifier (h1) with new data to form the training data for a new round, and train the second weak classifier (h2);

3) Combine the poorly-trained data of weak classifier (h1) and weak classifier (h2) with new data to form the training data for a new round, and train the third weak classifier (h3);

4) Repeat Steps 2-3 until an adequate number of weak classifiers have been attained.

5) Form a strong classifier through the weighted voting of all weak classifiers.

While the Boosting algorithm can enhance the generalization of algorithms, it has two weaknesses: first, it is necessary to know the lower limit of the learning performance of weak learners, which is difficult to do in reality; second, this method may lead to the problem of learners focusing so much attention on some data that are extremely difficult to train (maybe noise data), in follow-up learning, the algorithm

performance becomes unstable. Worse still, there are two difficult problems in the application of the Boosting algorithm, as follows:

1) How to adjust the training dataset, in order that weak learners are able to acquire the feature information that has not been acquired in future training.
2) How to combine all weak classifiers obtained from training to form a strong classifier.

B. ADABOOST ALGORITHM

AdaBoost (Adaptive Boosting) is one of the most representative Boosting algorithms, as proposed by Freund and Schapire in 1997. As its name suggests, according to the trained classifiers, AdaBoost can self-adjust weak classifiers after learning and is sensitive to noise data and outliers. In some tasks, it can efficiently resist overfitting. Even if an individual learner is weak, AdaBoost can converge into a strong classifier in the final integration, provided that the trained weak classifiers are better than the random classifiers.

Compared with the original Boosting algorithm, AdaBoost replaces random sampling with weighted sampling to train data. In this way, focus is placed on training data that are difficult to process, which renders training more targeted. Regarding the combination of classifiers, it replaces the average voting mechanism with a weighted voting mechanism when combining weak classifiers. By equipping the weak classifiers that are effective in classification with a higher weight, and equipping those that are ineffective in classification with a lower weight, it guarantees that the integrated weak classifiers will be effective in classification. To use AdaBoost, it is not necessary to know the learning performance of the weak algorithm in advance, and the classification accuracy of the integrated strong classifier depends on the classification accuracy of all weak classifiers. In this way, it is possible to tap the algorithm potential of weak classifiers. Basically, the concept of Adaoost can be described, as follows: the given training dataset $\mathrm{T} = \{(x_1, \mathrm{y}_1), (\mathrm{x}_2, \mathrm{y}_2) \ldots (\mathrm{x}_\mathrm{N}, \mathrm{y}_\mathrm{N})\}$, where $x \in X$, $\mathrm{X} \in R^n$, and $\mathrm{y_i}$ is a mark set $\{-1, +1\}$. Adaboost aims to acquire a series of weak classifiers from the training data, and then, combines these weak classifiers into a strong classifier. Moreover, its computational process can be described, as follows:

1) The weight distribution of the initial training data; each training sample is given the same initial weight: $1/N$, as shown in (16):

$$\mathrm{D}_1 = (w_{11}, w_{12} \ldots w_{1i} \ldots, w_{1N}), \quad w_{1i} = \frac{1}{N}, \quad i = 1, 2, \ldots, N \tag{16}$$

If sample data have been accurately classified, then the probability that they will be chosen in the construction of the next-round training set will become lower; conversely, if not accurately classified, then their weight will become higher.

2) In the case of $m = 1, 2, \ldots,$ M uses the training data with the weight distribution (D_m) in learning to obtain a basic binomial classifier, as shown in (17):

$$G_m(x) : x \to \{-1, +1\} \tag{17}$$

3) As shown in (18), the rate of a wrong classification of classifier ($G_m(x)$) in the training dataset is, as follows:

$$\varepsilon_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i) \tag{18}$$

4) As shown in (19), coefficient (α_m) of $G_m(x)$) is calculated, which refers to the voting weight of $G_m(x)$) in the final classier:

$$\alpha_m = \frac{1}{2} \log \frac{1 - \varepsilon_m}{\varepsilon_m} \tag{19}$$

According to the above equation, when $\varepsilon_m \leq 0.5$, $\alpha_m \geq 0$. Moreover, α_m increases with the decline of ε_m. This indicates that weak classifiers with a lower rate of wrong classification have higher voting weight in the final classifier.

5) As shown in (20) and (21), the weight distribution of the updated training datasets is, as follows:

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2} \ldots w_{m+1,i} \ldots, w_{m+1,N}) \tag{20}$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \; i = 1, 2, \ldots, N \tag{21}$$

The weight distribution of the training datasets is adjusted to increase the weight of the data that are inaccurately classified by $G_m(x)$, and decreases the weight of those that are accurately classified. In this way, AdaBoost can ''focus on'' the data that are difficult to process. As shown in (22), Z_m is a normalized factor, making D_{m+1} a probability distribution:

$$\mathrm{Z}_m = \sum_{i=1}^{N} w_{mi} \exp(-\alpha_m y_i G_m(x_i)) \tag{22}$$

6) As shown in (23), the linear combination of basic classifiers is established:

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x) \tag{23}$$

The final classifier is then obtained, as shown in (24):

$$G(x) = sign(f(x)) = sign(\sum_{m=1}^{M} \alpha_m G_m(x)) \tag{24}$$

C. SUMMARY

In this chapter, Section 3.1 introduces ensemble learning and its three strengths. Section 3.2 elucidates the concept of the Boosting algorithm, and lists its work processes and some weaknesses. Section 3.2 elaborates on the famous Boosting method, AdaBoost, and gives a detailed description of its computational procedure.

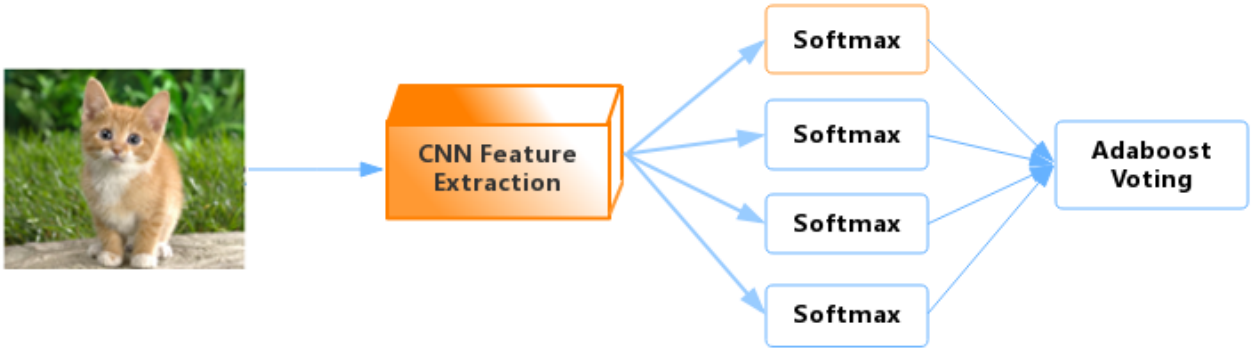

FIGURE 15. Proposed BoostCNN structure.

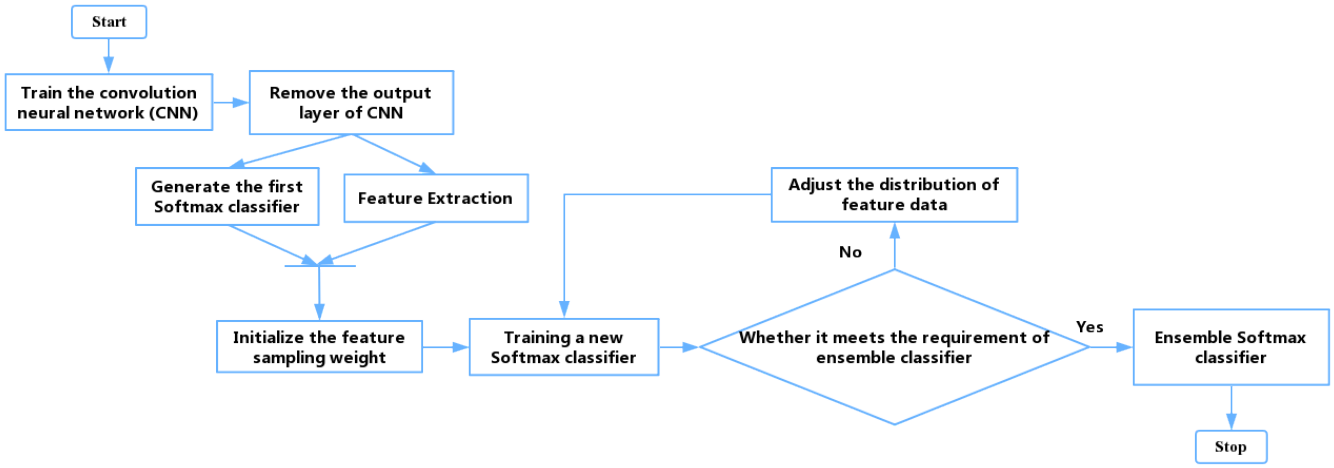

FIGURE 16. BoostCNN training procedure.

IV. BOOSTCNN METHOD

This chapter combines the convolution neural network with the AdaBoost algorithm to form the BoostCNN model. In a traditional integrated neural network, the neural network is usually taken as a base learner for ensemble learning; however, as it takes a long time to train an individual neural network, it has high total training cost, even though the integration of several neural networks can enhance the performance of individual neural networks. Therefore, BoostCNN is different from the traditional integrated neural network in the following aspects.

a) Use a convolution neural network for optimization training, in order to develop the optimal convolution neural network;
b) Remove the output layer of the trained convolution neural network and fix all the layers; extract the features and take them as the data for new training;
c) Use the extracted new training data and the Softmax classifier on the last layer of Step 1 as the base learners of AdaBoost for ensemble learning.

A. THE DESIGN OF BOOSTCNN

Figure 15 displays the complete BoostCNN structure. The orange module is the training stage of the convolution neural network, where BoostCNN is trained as a traditional convolution network. The blue module is the training stage of AdaBoost, where the convolution network is adopted to extract features, and its output layer network is taken as the base learner of the AdaBoost algorithm for the second learning.

In addition, the training procedure of BoostCNN is displayed in Figure 16.

B. THE TRAINING PROCESS OF CNN

The training stage of CNN is shown in Algorithm 1, where Adam [20] is adopted for learning the gradient reduction. By storing the historical gradient and learning the efficiency of each parameter, this algorithm can effectively prevent the difficult problems of a partial optimum, as well as the optimization of saddle points in the neural network of deep learning. This is currently one of the most popular optimization algorithms for training a deep neural network, and its procedure is described, as follows:

C. THE TRAINING PROCESS OF ADABOOST

In the training stage of AdaBoost, the trained convolution network is adopted to extract features. The first trained Softmax classifier is then taken as the initial base learner to adjust

Algorithm 1 Train CNN With the Learning Algorithm of Adam

Initialization:

The given dataset; dataset mark $Y = \{y^{(1)}, y^{(2)}, \ldots, y^{(n)}\}$; initial velocity is v; the size of random sampling is m; the training cycle is k; the convolution network is $f(x; w)$; the initial learning efficiency is α; the parameter of momentum decay is β_1; the parameter of learning efficiency decay is β_2; $\delta = 10^{-7}$.

Training:

For t < k

{ 1. "m" data are randomly selected as samples:

$$\{(x^{(1)}, y^{(1)}) \ldots\ldots (x^{(m)}, y^{(m)})\}$$

2. Calculate the gradient of the current sample data:

$$g = \frac{1}{m} \sum_{j=1}^{m} \frac{\partial \mathrm{L}(y^{(j)}, f(x^{(j)}; w))}{\partial \mathrm{w}}$$

3. Update the current velocity: $v = \beta_1 \cdot v + (1 - \beta_1)g$
4. Update the current learning efficiency: $r = \beta_2 \cdot r + (1 - \beta_2)g^2$
5. Update the frequency of training: t = t + 1

$$vb = \frac{v}{1 - \beta_1^t}, \quad rb = \frac{r}{1 - \beta_2^t}$$

6. Update parameter: $w = w - \frac{\alpha}{\sqrt{rb} + \delta} vb$ }

the data distribution probability of feature extraction. The specific training process is, as follows:

1) Remove the output layer of the convolution network and extract the features from the original training data;
2) Take the trained Softmax classifier of the output layer of the convolution network as the first base learner of AdaBoost; store the training accuracy; adjust the data sampling weights of the feature extraction data;
3) Use the new data sampling weights for the iterative training of the next base learner and store the training accuracy;
4) Conduct the iterative training of several Softmax classifiers;
5) Fix all Softmax classifiers and use their training accuracies as the final voting weights; conduct the integrated output of AdaBoost.

D. SUMMARY

This chapter takes the convolution neural network as the image feature extraction model, and the Softmax classifier of its output layer is adopted as the base learner for AdaBoost ensemble learning, in order to improve the identification performance of the convolution neural network, while reducing the consumption of memory resources and shortening the training cycle as much as possible.

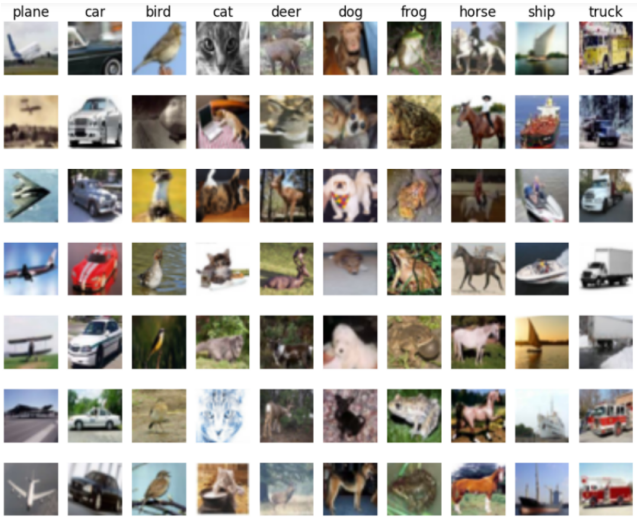

FIGURE 17. CIFAR-10 image dataset.

V. SIMULATION EXPERIMENT AND ANALYSIS

To analyze the performance of the BoostCNN model, this chapter uses the four-layered convolution network and Softmax classifier as the feature learning architecture, and adopts the AdaBoost algorithm to generate several Softmax classifiers as the output. Moreover, the CIFAR-10 dataset is employed to test the performance of the network architecture.

A. CIFAR-10 DATASET

Developed by Krizhevsky *et al.* from the Hinton team, the CIFAR-10 dataset is an image dataset whose 60,000 colorful images (32 ∗ 32) are classified into 10 categories. Specifically, 50,000 images are classified as training data, while 10,000 images are classified as testing data. As shown in Figure 17, the 10 categories are plane, car, bird, cat, deer, dog, frog, horse, ship, and truck, and each category includes 6,000 images.

This paper adopts the Python dataset. To facilitate the application of this dataset, a two-dimensional array (50000, 3072) is employed to represent the dataset. The horizontal lines of an array indicate the number of data, while the vertical rows refer to the features of data; hence, each line stores an RGB image (32 ∗ 32). At the same time, "3072" represents that each image has 1,024 dimensions of the red channel, 1,024 dimensions of the green channel, and 1,024 dimensions of the blue channel, and each dimension of data indicates a pixel value.

B. EXPERIMENTAL FACILITY

Table 1 lists the hardware and software of the machine. The currently popular Tensorflow deep learning library is adopted for training.

C. THE DEPLOYMENT OF THE CNN MODEL

Table 2 and Table 3 list the deployment of the CNN model and the configuration of Adaboost, respectively.

TABLE 1. Hardware and software of computer.

Item	Content
Processor	Intel(R) core(TM) i7-6700HQ CPU
GPU	NVIDIA GeForce GTX 960M
Memory	8G
Operating System	Windows 10
Tensorflow	TensorFlow 1.0
Python	Python 3.5
Cuda	cuda 8.0

TABLE 2. Specification of CNN configuration.

Input: 500000*3072		
Hiden1 Layer	conv	Size 5*5; quantity: 64; method: same
	ReLU	Max(0,x)
	Max Pooling	Size: 3*3; stride:2
	Batch Norm	alpha=0.001 / 9.0, beta=0.75
Hiden2 Layer	conv	Size: 5*5; quantity: 64; method: same
	ReLU	Max(0,x)
	Max Pooling	Size: 3*3, stride:2
	Batch Norm	alpha=0.001 / 9.0, beta=0.75
Hiden3 Layer	Full connect	Weight size: [1228, 384]
	ReLU	Max(0,x)
	Dropout	Probability of activation: 0.5
Hiden4 Layer	Full connect	Size of weight: [384, 192]
Output Layer	Softmax	Size of weight: [192, 10]

TABLE 3. Configuration of adaboost.

Input	Use the feature extraction data of the convolution network: [50000,192]
Softmax1	Size of weight: [192, 10]
Softmax2	Size of weight: [192, 10]
Softmax3	Size of weight: [192, 10]
Softmax4	Size of weight: [192, 10]
Softmax5	Size of weight: [192, 10]
Softmax6	Size of weight: [192, 10]
Softmax7	Size of weight: [192, 10]
Output	Results of weight voting of the categories

TABLE 4. Experimental comparison of CIFAR-10 testing datasets.

Classifier	Accuracy of Testing (%)
Softmax	35.5
AdaBoost+ Softmax	52.3
CNN+Softmax	85.3
CNN+AdaBoost (this study)	88.4

D. COMPARISON OF RESULTS

As shown in Table 4, the CIFAR-10 dataset is adopted to test the performance of Softmax, AdaBoost + Softmax, CNN + Softmax, and CNN + AdaBoost.

According to Table 4, when the original images are learned without feature extraction, the testing accuracy of the Softmax classifier is only 35.5%; after AdaBoost ensemble learning, while accuracy increases to 52.3%, the overall performance remains low. After the deep convolution network is adopted for feature extraction, the accuracy of testing significantly increases to 85.3%. The CNN + AdaBoost architecture, as adopted in this paper, can increase the accuracy of the original CNN by 3%, and its classification accuracy can reach 88.4%.

E. SUMMARY

This chapter elaborates on the testing process of BoostCNN, where the CIFAR-10 image dataset is employed to test and analyze the learning effects of BoostCNN, Softmax, CNN, and AdaBoost. According to the experimental results, BoostCNN can enhance CNN performance by 3%.

VI. CONCLUSION

This study focuses on how to combine a convolution neural network with AdaBoost to enhance the image identification performance of the learning algorithms. After the convolution neural network is trained into the deep feature extraction model, and the original images are converted to acquire the deep features, AdaBoost is used for ensemble learning. The conclusions of this paper are, as follows:

1) After the feature extraction of the deep convolution neural network, the original image data are fully abstracted, thus, the traditional learning algorithms can also effectively learn highly complicated image data.
2) Through the comparative experiments of the CIFAR-10 image dataset, AdaBoost can generally enhance the performance of base learners by 3%.

The model of this study does have some limitations.

1) The identification performance of the model is largely dependent on the first stage, meaning the image feature extraction of the convolution neural network. It is relatively passive in learning abstract features.
2) While the model can effectively improve accuracy, it still has high time costs.

The convolution neural network is one of the most successful machine learning models to emerge from the deep learning industry in recent years. As the hardware of computers and available datasets advance, the convolution neural network has become increasingly larger, and its performance has become better. At the same time, the requirements of the infrastructure for the training of deep learning models have become stricter, and training has become increasingly difficult. Even if a high-performance computing device is available, it may still take days and even weeks to train a deep learning model. Therefore, how to reduce the time and resource costs for the training of a deep learning model is an important research issue. This paper separates the feature extraction layer and the output layer classifiers of the deep convolution network from each other, and replaces the output layer with the AdaBoost algorithm in an attempt to combine deep learning with traditional algorithms. Likewise,

a trained deep convolution network can be integrated with Bagging or SVM to enhance the performance of algorithms, while saving training cost.

REFERENCES

[1] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.
[2] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, 2011.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[4] H. Zou, J. Zhu, S. Rosset, and T. Hastie, "Multi-class AdaBoost," *Statist. Interface*, vol. 2, pp. 349–360, 2009.
[5] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[6] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, Apr. 1980.
[7] F. Lauer, C. Y. Suen, and G. Bloch, "A trainable feature extractor for handwritten digit recognition," *Pattern Recognit.*, vol. 40, no. 6, pp. 1816–1824, Jun. 2007.
[8] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 609–616.
[9] W. Ouyang *et al.* (2014). "DeepID-Net: Multi-stage and deformable deep convolutional neural networks for object detection." [Online]. Available: https://arxiv.org/abs/1409.3505
[10] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 1026–1034.
[11] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 10, pp. 993–1001, Oct. 1990.
[12] M. Steyvers and J. B. Tenenbaum, "The Large-scale structure of semantic networks: Statistical analyses and a model of semantic growth," *Cognit. Sci.*, vol. 29, no. 1, pp. 41–78, Jan. 2005.
[13] J. Nagi *et al.*, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *Proc. IEEE Int. Conf. Signal Image Process. Appl. (ICSIPA)*, Nov. 2011, pp. 342–347.
[14] S. Zubair, F. Yan, and W. Wang, "Dictionary learning based sparse coefficients for audio classification with max and average pooling," *Digit. Signal Process.*, vol. 23, no. 3, pp. 960–970, May 2013.
[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[17] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
[18] T. G. Dietterich, "Ensemble methods in machine learning," in *International Workshop on Multiple Classifier Systems* (Lecture Notes in Computer Science), vol. 1857. 2000, pp. 1–15.
[19] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proc. Eur. Conf. Comput. Learn. Theory*, 1995, pp. 23–37.
[20] D. P. Kingma and J. Ba. (2014). "Adam: A method for stochastic optimization." [Online]. Available: https://arxiv.org/abs/1412.6980

SHIN-JYE LEE received the M.Sc. (Eng.) degree from the Department of Computer Science, The University of Sheffield, U.K., in 2001, the M.Phil. degree from the Judge Business School, University of Cambridge, U.K., in 2012, and the Ph.D. degree from the School of Computer Science, The University of Manchester, U.K., in 2011. He was with the National Pilot School of Software, Yunnan University, China, and he also made his academic career in Poland, in 2012. In addition, he also had practical experiences at Fujitsu and Microsoft, from 2002 to 2005. He is currently an Assistant Professor with the Institute of Technology Management, National Chiao Tung University, Taiwan. His research interests primarily comprise machine learning, computational intelligence and decision support system, operational research, and technology policy, especially for the climate change issues and energy prediction.

TONGLIN CHEN is currently pursuing the master's degree with the National Pilot School of Software, Yunnan University, China. His research interests primarily comprise machine learning, artificial neural network, support vector machine, and optimization methods.

LUN YU is currently pursuing the master's degree with the National Pilot School of Software, Yunnan University, China. Her research interests primarily comprise machine learning and convolutional neural network.

CHIN-HUI LAI received the M.S. and Ph.D. degrees from the Institute of Information Management, National Chiao Tung University, in 2004 and 2010, respectively. She was a Post-Doctoral Fellow with the Institute of Information Management, National Chiao Tung University,, in 2010. She is currently an Associate Professor with the Department of Information Management, Chung Yuan Christian University, Taiwan. Her research interests include data mining, text mining, machine learning, social network analysis, and recommender systems.