micromachines

MDPI

Article

Nonlinear Hyperparameter Optimization of a Neural Network in Image Processing for Micromachines

Mingming Shen [1,2], Jing Yang [1,3], Shaobo Li [1,3,*], Ansi Zhang [1,3] and Qiang Bai [1]

[1] School of Mechanical Engineering, Guizhou University, Guiyang 550025, China; 15985146314@163.com (M.S.); jyang23@gzu.edu.cn (J.Y.); zhangas@gzu.edu.cn (A.Z.); 18798824036@163.com (Q.B.)
[2] School of Mechanical & Electrical Engineering, Guizhou Normal University, Guiyang 550025, China
[3] State Key Laboratory of Public Big Data, Guizhou University, Guiyang 550025, China
* Correspondence: lishaobo@gzu.edu.cn

Abstract: Deep neural networks are widely used in the field of image processing for micromachines, such as in 3D shape detection in microelectronic high-speed dispensing and object detection in microrobots. It is already known that hyperparameters and their interactions impact neural network model performance. Taking advantage of the mathematical correlations between hyperparameters and the corresponding deep learning model to adjust hyperparameters intelligently is the key to obtaining an optimal solution from a deep neural network model. Leveraging these correlations is also significant for unlocking the "black box" of deep learning by revealing the mechanism of its mathematical principle. However, there is no complete system for studying the combination of mathematical derivation and experimental verification methods to quantify the impacts of hyperparameters on the performances of deep learning models. Therefore, in this paper, the authors analyzed the mathematical relationships among four hyperparameters: the learning rate, batch size, dropout rate, and convolution kernel size. A generalized multiparameter mathematical correlation model was also established, which showed that the interaction between these hyperparameters played an important role in the neural network's performance. Different experiments were verified by running convolutional neural network algorithms to validate the proposal on the MNIST dataset. Notably, this research can help establish a universal multiparameter mathematical correlation model to guide the deep learning parameter adjustment process.

Keywords: deep neural network; hyperparameters; multiparameter mathematical correlation model; image processing

Citation: Shen, M.; Yang, J.; Li, S.; Zhang, A.; Bai, Q. Nonlinear Hyperparameter Optimization of a Neural Network in Image Processing for Micromachines. *Micromachines* **2021**, *12*, 1504. https://doi.org/10.3390/mi12121504

Academic Editor: Young Min Song

Received: 20 October 2021
Accepted: 27 November 2021
Published: 30 November 2021

1. Introduction

Deep neural networks are widely used in the field of image processing for micromachines. For example, a deep neural network model can be used in detecting the 3D shapes of droplets online in the process of microelectronic high-speed dispensing or droplets formed by microfluidic chips [1]. The convolutional neural network (CNN) has been proven to have better performance in hyperspectral image (HSI) classification than traditional methods, while HSI classification of remote sensing makes full use of high-altitude detection equipment [2]. In particular, neural networks (DNN) have recently proven their effectiveness when applied to image recognition to extract the meaningful information from sensors in a smart tactile sensing system of micromachines [3]. The outstanding performance of each of the above deep neural networks is inseparable from the excellent design of the system architecture and the elaborate selection of hyperparameter values. Hyperparameters, such as the learning rate (lr), batch size (m), convolution kernel size (ke), and dropout rate (q), determine the performance of a deep neural network model, which has a significant impact on the deep-learning-based system of micromachines. There is an urgent need for a systematic, mathematical study of hyperparameters in deep neural

networks and to explain how to set the hyperparameters to get the best performance of the model.

However, choosing the right value for a given hyperparameter is very tricky, because doing so depends not only on the programmer's experience level, but also on the ability of the model to learn from each round of value experiments. In general, the strategy of combining a grid search and a manual search is adopted for hyperparameter configuration [4–6]. The manual search method has difficulty reproducing the obtained experimental results, and the grid search strategy is only suitable for cases involving a small number of hyperparameters in practice, and is performed utilizing an expensive trial-and-error approach. Although random search [7] has been proposed, and researchers can identify good hyperparameter values with accuracy by random sampling from the search space, random search has the possibility of performing repeated searches with the same hyperparameters. The Bayesian-optimization-based algorithms improve the random-search-based optimization process, and can find better hyperparameters more quickly [8–11]. However, these methods can only perform static hyperparametric searches. Hyperparameter optimization algorithms are proposed based on reinforcement learning, such as the particle swarm optimization method [12,13], multiparameter optimization method [14,15], and dynamic optimization method [16–18]. These algorithms have been proven to be beneficial for optimizing structural parameters. However, hyperparameters such as *lr*, *q*, *m*, and *ke* still need to be manually set by the user. Determining the optimal values of these hyperparameters has always been a key objective in the field of deep learning research. At present, the configuration and optimization of hyperparameters mostly rely on experiments and experience, and researchers generally select hyperparameter values within certain ranges based on others' experiences. This approach lacks a systematic description of the effects of multiple hyperparameters on the performances of deep learning models based on mathematical principles and convolution theory.

Here, after the Introduction, the authors present a review of the literature investigating the relationship between hyperparameter tuning and model performance. These studies relate to the interpretability of deep CNNs, the optimizer of the deep neural network model and commonly used hyperparameters. The mathematical equations between individual hyperparameters (*lr*, *m*, *q*, and *ke*) and model performance are derived separately based on the previous work. Based on this, the authors attempt to integrate the four hyperparameters and derive mathematical relationships to account for the interactive effects of these hyperparameters on model performance. After completing the mathematical derivation, the authors first conduct single-hyperparameter cross-validation experiments on three models while considering the effects of individual hyperparameters. Then, they select the optimal hyperparameter combination based on the above-mentioned results and conduct multiparameter validation experiments using the three models again. Finally, the conclusions are drawn in the last part. Combining mathematical derivation and experimental validation, the authors want to give a systematic, mathematical study to hyperparameters in deep neural networks and establish a universal multiparameter mathematical correlation model to guide the hyperparameters adjustment process.

The main contributions of this paper are as follows:

1. The correlations among the *lr*, *m*, *q*, and *ke* are explained through a theoretical derivation. A generalized mathematical model is established to guide the parameter adjustment process for the examined deep learning model.

2. The effective influences of *lr*, *m*, *q*, and *ke* on the performance (the convergence of the accuracy rate, the cross-entropy loss, and the running time) of the deep learning model are confirmed by performing multiple validations using different hyperparameter values and different architectures. The value range of each parameter when the model reaches its relatively optimal state is obtained, which provides suggestions for the hyperparameter designs of deep learning network architectures.

3. The universal multiparameter mathematical correlation model gives some clarity to the process performed by the CNNs and their black box by mathematical derivation and experiment validation.

2. Related Work

2.1. The Interpretability of Deep CNNs

A deep learning model is called a "black box" because of its opaque learning and prediction processes, and the black box problem of deep learning has become a hot and difficult issue in artificial intelligence (AI) research [19–23]. Many methods have been proposed to explain deep learning models. For example, the local interpretable model-agnostic explanation (LIME) [24] is a method that is often used to simulate the output of a black box model with the advantages of form simplicity, easy modeling, and good interpretability, but it has difficulty dealing with models with complex features. The decision tree model [25] is a machine-learning-based self-explanatory model with a tree structure composed of internal nodes and leaf nodes, and it can quantify and explain the prediction logic of a CNN at the semantic level. Category activation approaches [26] based on the decision tree method have difficulty providing an understanding of the decision rules of trees as the tree depth and number of leaves increase. Class activation mapping (CAM) [27] is a method that can explain CNNs and identify the positions of classified objects. However, the CAM technique has a high model training cost, which greatly limits the application scenarios of the model. To solve the problems of the CAM method, the Grad-CAM [28] method was proposed, which is suitable for a variety of CNN models, and does not change the network structure or require retraining. Grad-CAM can generate class activation heat maps for a network on the basis of maintaining the original network structure, but cannot solve the gradient saturation problem. For this reason, the integrated gradient method [29] and deep-lift method [30] were proposed. Although these methods can explain a deep learning model to a certain extent or from different perspectives, they do not systematically explain the influences of hyperparameters (i.e., the learning rate, batch size, dropout rate, and convolution kernel size) on the deep learning performance of the model in principle.

2.2. The Optimizer of the Deep Convolution Neural Network Model Refers to Key Hyperparameters

Gradient methods (such as the gradient descent method (GD) [31], stochastic gradient descent method (SGD) [32], and small-batch stochastic gradient descent method (M-BGD) [33]) are the most commonly used optimization algorithms for deep learning models, and many subsequent algorithms have been developed accordingly to expand such methods. For example, the momentum method [34] is a new method for SGD to accelerate and suppress the correlation direction; this approach has the advantages of increasing the stability and speeding up the learning process of SGD, and the disadvantage is that the inertia of the system oscillates near the optimal solution when the learning speed is fixed. The Nesterov accelerated gradient (NAG) method [35,36] is an improvement of the momentum method that takes the cumulative adjustment of momentum into account, avoids moving too fast, and simultaneously improves the sensitivity of the developed model. Adaptive gradient (AdaGrad) [37] is an optimization algorithm that can adaptively adjust the learning rate of a model according to its gradient during the training process after setting the initial learning rate, but the disadvantage of this method is that as the cumulative squared gradient sum increases during the training period, the learning rate continues to shrink and eventually becomes infinitely small, which is not conducive to approaching the optimal solution in the later stage of training. Adaptive delta (AdaDelta) and root-mean-square propagation (RMSProp) were proposed to solve the problem of the AdaGrad learning rate monotonically decreasing [38]. AdaDelta uses the root-mean-square error of a given parameter to approximate the update process; RMSProp divides the learning rate by the exponentially decayed average of the squared gradient, then uses the learning rate and associates it with the gradient. They both use an attenuation coefficient to

control the amount of gradient information considered, and may encounter saddle points during the training process. For the saddle point problem encountered during training, evolutionary stochastic gradient descent (ESGD) [39] was proposed for designing a more suitable adaptive learning rate method using a Hessian negative eigenvalue, and the convergence of this method was determined to be the same or faster than that of RMSProp. Adaptive moment estimation (Adam) [40] is another method for the adaptive calculation and adjustment of hyperparameters. This method not only stores the exponential decay average of the past gradient squares, but is also suitable for nonconvex optimization, noisy data, and sparse gradient problems in high-dimensional spaces. Adaptive max pooling (AdaMax) limits the upper bound of the learning rate and does not consider noise deviation; its parameter update method is simpler than that of Adam. Nesterov-accelerated adaptive moment estimation (Nadam) [41] is an optimization method that combines the Adam and NAG algorithms. The AdaptAhead [42] method is based on the Nesterov and RMSProp methods; it introduces first- and second-order attenuation rate parameters, initializes the model weights, and has a faster convergence speed than those of the methods on which it is based. In addition, in [43], an optimizer that combined dropout and SGD was used to improve the CNN algorithm, thereby improving the feature recognition rate and reducing the time cost of the examined CNN. In summary, hyperparameters such as *lr* and *m* are closely related to optimization algorithms. In addition, the hyperparameters of a deep learning model include not only the hyperparameters related to the corresponding optimization algorithm, but also the hyperparameters of the network structure, such as the size of the convolution kernel and the selection of the regularization method [44]. At present, these hyperparameters are usually selected based on experiments and previous experience. Optimizing hyperparameters is still the main problem with respect to training deep learning models. Therefore, studying the correlations between various hyperparameters and establishing a universal mathematical model is of great significance regarding the construction of deep learning models.

2.3. Commonly Used Hyperparameters

The *q*, *m*, *lr*, and *ke* are important parameters of the deep learning model, and their design and settings directly affect the performance of the deep learning model. The use of *q* can adjust the neural network being trained, and these random modifications to the network structure are thought to avoid neuronal coadaptation by making it impossible for two consecutive neurons to be completely interdependent [45–47]. The selection of *m* affects the quality of confidence interval estimators, and there is a certain relationship between sample size and optimal batch; a small batch is used to approximate the gradient of the loss function relative to the parameter [48]. The training is carried out in steps, and we consider a small batch in each step. An *lr* that is too small will cause the training algorithm to converge slowly, while an *lr* that is too large will cause the training algorithm to diverge [32]. Researchers [35] found that blindly superimposing the number of network layers is not an effective way to design an efficient and deeper network. The problem of designing a deep CNN under certain restricted conditions can be transformed into an optimization problem under restricted conditions, which is important. One of the influencing factors is the *ke*. By decomposing the convolution with an appropriate *ke*, more decoupling parameters can be obtained, and training can be accelerated. However, the current research only focuses on the experimental verification of a single parameter or two parameters. Based on this, we systematically described the impact of the four parameters of *q*, *m*, *lr*, and *ke* on the deep learning model; derived a mathematical formula for the relationship between the weights and bias of the four hyperparameters and the loss function of the deep learning model; and then verified the relative optimal value range of each hyperparameter through experimental verification.

3. Mathematical Deduction of the Model Optimization Process Considering a Single Hyperparameter

3.1. Basic Principles of a Deep Neural Network

A deep neural network is a neural network composed of many hidden layers, and its basic structure consists of an input layer, an output layer, and multiple hidden layers. Therefore, a deep neural network can explain the relationship between the input and output of its hidden layer by a linear model, and the input of the latter layer of the neural network is the output of the previous layer.

We make the following two assumptions: (1) The training dataset is a dataset X containing K categories, where a single sample X is an eigenvector with i neurons, $\vec{x} = (x_1, x_2, \cdots x_i)$ and (2) the deep learning model has L hidden layers, denoted as $l = (1, 2, \ldots, L)$, $z^{(l)}$ represents the input vector of the l layer, and $y^{(l)}$ represents the output vector from the l layer (where $y^{(0)} = x$ is the original input).

Therefore, when the input is a single sample from a given category, the model input function and output function of each layer in the deep neural network are related to the weight and bias relations in the network, as shown in Equation (1):

$$\begin{cases} y^{(0)} = z^{(0)} = \vec{x} \\ y^{(1)} = f\left(z^{(1)}\right) = f\left(\sum\limits_{i=1} w_i^{(1)} \cdot x_i^{(1)} + b_i^{(1)}\right) = f\left(\sum\limits_{i=1} w_i^{(1)} \cdot y^{(0)} + b_i^{(1)}\right) \\ y^{(2)} = f\left(z^{(2)}\right) = f\left(\sum\limits_{i=1} w_i^{(2)} \cdot x_i^{(2)} + b_i^{(2)}\right) = f\left(\sum\limits_{i=1} w_i^{(2)} \cdot y^{(1)} + b_i^{(2)}\right) \\ \qquad\qquad \cdot \\ \qquad\qquad \cdot \\ \qquad\qquad \cdot \\ y^{(l)} = f\left(z^{(l)}\right) = f\left(\sum\limits_{i=1} w_i^{(l)} \cdot x_i^{(l)} + b_i^{(l)}\right) = f\left(\sum\limits_{i=1} w_i^{(l)} \cdot y^{(l-1)} + b_i^{(l)}\right) \\ y^{(l+1)} = f\left(z^{(l+1)}\right) = f\left(\sum\limits_{i=1} w_i^{(l+1)} \cdot x_i^{(l+1)} + b_i^{(l+1)}\right) = f\left(\sum\limits_{i=1} w_i^{(l+1)} \cdot y^{(l)} + b_i^{(l+1)}\right) \end{cases} \tag{1}$$

where $x_i^{(l)}$ represents the input of the l layer; $w_i^{(l)}$ and $b_i^{(l)}$ are the weight and bias of the l layer of $x_i^{(l)}$; respectively, and $f(Z)$ represents the activation function.

Then, when the input is a single sample of K categories, the input function and output function of each layer of the deep neural network are expressed as Equation (2):

$$\begin{cases} y_k^{(0)} = z_k^{(0)} = \vec{x} \\ y_k^{(1)} = f\left(z_k^{(1)}\right) = f\left(\sum\limits_{i=1} w_{k,i}^{(l)} \cdot x_{k,i}^{(1)} + b_{k,i}^{(1)}\right) = f\left(\sum\limits_{i=1} w_{k,i}^{(1)} \cdot y_k^{(0)} + b_{k,i}^{(1)}\right) \\ y_k^{(2)} = f\left(z_k^{(2)}\right) = f\left(\sum\limits_{i=1} w_{k,i}^{(2)} \cdot x_{k,i}^{(2)} + b_{k,i}^{(2)}\right) = f\left(\sum\limits_{i=1} w_{k,i}^{(2)} \cdot y_k^{(1)} + b_{k,i}^{(2)}\right) \\ \qquad\qquad \cdot \\ \qquad\qquad \cdot \\ \qquad\qquad \cdot \\ y_k^{(l)} = f\left(z_k^{(l)}\right) = f\left(\sum\limits_{i=1} w_{k,i}^{(l)} \cdot x_{k,i}^{(l)} + b_{k,i}^{(l)}\right) = f\left(\sum\limits_{i=1} w_{k,i}^{(l)} \cdot y_k^{(l-1)} + b_{k,i}^{(l)}\right) \\ y_k^{(l+1)} = f\left(z_k^{(l+1)}\right) = f\left(\sum\limits_{i=1} w_{k,i}^{(l+1)} \cdot x_{k,i}^{(l+1)} + b_{k,i}^{(l+1)}\right) = f\left(\sum\limits_{i=1} w_{k,i}^{(l+1)} \cdot y_k^{(l)} + b_{k,i}^{(l+1)}\right) \end{cases} \tag{2}$$

where $x_{k,i}^{(l)}$ represents the input sample from the k category in the l layer; $w_{k,i}^{(l)}$ and $b_{k,i}^{(l)}$ are the weight and the bias of the l layer of class k, respectively; and $f(Z)$ represents the activation function.

It can be seen from Equations (1) and (2) that in deep learning, bias and weight have decisive influences on the performance of each layer of the deep neural network model, regardless of the model input or output, when activation functions have been selected. Hyperparameters such as the learning rate, batch size, kernel size, and dropout rate affect

the update processes of the network weights and biases. Therefore, this section mainly focuses on a correlation analysis regarding the influence of four hyperparameters, the kernel size, dropout rate, batch size, and learning rate, on the performance of a deep learning model.

3.2. Considering the Convolution Kernel Size

A CNN is a deep neural network using a convolutional layer. To extract data features (or carry out feature learning), the weight and bias of the network are shared for neurons in the same layer. Research [49] shows that the following factors determine the size of the output feature map after convolution: the input feature map size, the convolution kernel size, the stride size, and the padding size. The relationship between the output size and determinants is shown in Equation (3):

$$O = \frac{I - ke + 2P}{S} + 1 \tag{3}$$

In Equation (3), I is the size of the input image, S is the stride size, P is the padding size, and O is the output image size.

Equation (3) shows that the convolution kernel size is one of the main parameters affecting the size of the output feature map of each convolution layer.

When other parameters remain unchanged, the convolution kernel size not only affects the output image size of the convolutional layer, but also affects the number of weights and the number of total parameters of the convolutional layer. At this time, the relationship between the number of parameters of the convolutional layer and the convolutional kernel size can be expressed by Equation (4):

$$\begin{cases} w_C = ke^2 \times ch \times N \\ B_C = N \\ P_C = W_C - B_C \end{cases} \tag{4}$$

where W_C and B_C are the numbers of weights and biases of the convolutional layer, Pc is the total number of parameters of the convolutional layer, N is the predefined number of cores, and ch is the number of input image channels.

At the same time, the size of the convolution kernel also affects the number of output parameters of the fully connected layer. Since the fully connected layer has a characteristic that the length of its output vector is equal to the number of neurons in the layer, the number of parameters is the sum of all weights and biases. The fully connected layer is divided into two cases; namely, the fully connected layer of the previous layer connected to the convolutional layer and the fully connected layer of the whole layer connected to the fully connected layer. The number of parameters is calculated as follows for the two cases:

(a) The calculation process for the parameters of the fully connected layer connected to the convolutional layer is shown in Equation (5):

$$\begin{cases} W_{cf} = O'^2 \times N \times F \\ B_{cf} = F \\ P_{cf} = W_{cf} - B_{cf} \end{cases} \tag{5}$$

where W_{cf} and B_{cf} are the numbers of weights and biases of the fully connected layer connected to the convolutional layer, respectively; O' is the output image size of the previous convolutional layer; F is the number of neurons in the current fully connected layer; and P_{cf} is the total number of parameters in the current fully connected layer.

(b) The calculation process for the parameters of the fully connected layer connected to another fully connected layer is shown in Equation (6):

$$\begin{cases} W_{ff} = F_{-1} \times F \\ B_{ff} = F \\ P_{ff} = W_{ff} - B_{ff} \end{cases} \quad (6)$$

where W_{ff} and B_{ff} are the numbers of weights and biases of the current fully connected layer, respectively; F_{-1} is the number of neurons in the previous fully connected layer; and P_{ff} is the total number of parameters in the current fully connected layer.

The convolutional kernel size is an important parameter for a CNN that affects the number of parameters involved in the calculation of the neural network.

3.3. Considering the Dropout Rate

Dropout is proposed to solve the overfitting problem of complex feedforward neural networks when training small datasets, and the dropout rate is used as a hyperparameter for training deep neural networks. The principle of dropout is to temporarily discard the neural network unit from the network in accordance with a certain probability during the training process of the deep learning network [50,51].

For l layer, r is a vector of independent Bernoulli random variables that obeys the Bernoulli distribution:

$$\begin{cases} P_r(r^{(l)} = 1) = 1 - q \\ P_r(r^{(l)} = 0) = q \end{cases} \quad (7)$$

It is assumed that dropout is adopted to discard the neurons in the neural network with a probability of q. Suppose that the l layer adopts the neural network model of dropout; then, the mathematical expressions of the output and input after refinement with dropout are shown in Equation (8):

$$\begin{cases} \widetilde{y}^{(l)} = r^{(l)} * y^{(l)} \\ z^{(l+1)} = w_i^{(l+1)} \widetilde{y}^{l} + b_i^{(l+1)} = w_i^{(l+1)} r^{(l)} * y^{(l)} + b_i^{(l+1)} \\ y^{(l+1)} = f(z^{(l+1)}) \end{cases} \quad (8)$$

For a sample from a dataset with k categories, according to Equation (4), the output of the corresponding layer is:

$$\begin{cases} \widetilde{y}_k{}^{(l)} = r^{(l)} * y_k{}^{(l)} \\ z_k{}^{(l+1)} = w_{k,i}^{(l+1)} \widetilde{y}_k^{(l)} + b_{k,i}^{(l+1)} = w_{k,i}^{(l+1)} r^{(l)} * y_k{}^{(l)} + b_{k,i}^{(l+1)} \\ y_k^{(l+1)} = f(z_k^{(l+1)}) \end{cases} \quad (9)$$

where $\hat{y}^{(l)}$ is the vector output from the layer after being refined by dropout.

However, in the testing phase, the weight is scaled; that is, the neural network obtained during the test is used without dropout.

3.4. Correlation Analysis Regarding the Batch Size and Learning Rate in the Binary and Multiclassification Case

The learning rate and batch size directly determine the weight update of the deep learning model, and affect the most important parameters with respect to model performance (convergence) [52,53]. The deep learning model process mainly includes convolution, dropout processing, a fully connected layer, and finally a classification model function for achieving target classification. According to the given sample label categories, we can divide the target classification problem into binary classification (the sample label has only two categories) and multiclass classification (the sample label has more than two categories). According to the target classification situation, the applicable classification model function is different.

Binary classification divides a sample into two categories (Class 1 and Class 2). Assuming that Class 1 is a positive sample, the probability that the sample is of the positive

type is expressed as the probability of the sample belonging to Class 2: $1 - P(C_1|x)$, and the classification model function is based on the sigmoid function. That is, the activation function f at this time is the sigmoid function. In a multilayer CNN model with an L-type hidden layer, the classification link is located in the output layer, and the input and output mathematical expressions are shown in Equation (10):

$$\begin{cases} z^{(l+1)} = \sum_{i=1} w_i{}^{(l+1)} \cdot x_i^{(l+1)} + b_i^{(l+1)} \\ f(z^{(l+1)}) = P(C_1 \Big| x) \\ f(z^{(l+1)}) = \sigma(z^{(l+1)}) = \frac{1}{1+e^{-z^{(l+1)}}} \end{cases} \tag{10}$$

where $P(C_1|x)$ represents the probability of a positive sample, and $P(C_1|x) \epsilon (0, 1)$.

According to the likelihood estimation and the chain rule, when the type of sample label is positive (when $P(C_1|x) = 1$), the loss function is 0. If the type of sample label is negative (when when $P(C_1|x) = 0$), the loss function is 1. According to Equation (10), the loss function is a function of the network weight and bias, and the loss function can be used to find the best w^* and b^* for minimization purposes.

Let $f_{w,b}(x) = f(z^{(l+1)}) = P(C_1 \Big| x)$; for a single sample, its loss function can be expressed as:

$$\text{Cost}(f_{w,b}\,(x), y) = \begin{cases} -\log(f_{w,b}(x)) \text{ if } \hat{y} = 1 \\ -\log(1 - f_{w,b}\,(x)) \text{ if } \hat{y} = 0 \end{cases} \tag{11}$$

In Equation (11), $\hat{y}$ is the target label, $\hat{y} = 1$ represents the sample belonging to Class 1, and $\hat{y} = 0$ represents the sample belonging to Class 2.

By writing the combined Equation (11) into a single equation, the equation of the cross-entropy loss function can be written as:

$$\text{Cost}(f_{w,b}(x), \hat{y}) = -[(\hat{y}) \log(f_{w,b}(x)) + (1 - \hat{y}) \log(1 - f_{w,b}(x))] \tag{12}$$

For a sample with batch size = m, its loss function is the mean value of the loss function jointly caused by x and y:

$$J(w, b) = \frac{1}{m} \sum_{j}^{m} \text{Cost}(f_{w,b}(x^{(j)}), \hat{y}^{(j)}) \tag{13}$$

Furthermore, Equation (13) is substituted into Equation (12) to obtain:

$$J(w, b) = -\frac{1}{m} \sum_{j=1}^{m} [(\hat{y}^{(j)}) \log(f_{w,b}(x^{(j)})) + (1 - \hat{y}^{(j)}) \log(1 - f_{w,b}(x^{(j)}))] \tag{14}$$

According to Equation (10), Equation (14) can be written as:

$$J(w, b) = -\frac{1}{m} \sum_{j=1}^{m} [\hat{y}^{(j)} \ln f_{w,b}(x^{(j)}) + (1 - \hat{y}^{(j)}) \ln(1 - f_{w,b}(x^{(j)}))] \tag{15}$$

The partial derivative of $J(w,b)$ with respect to w_i is calculated according to the gradient descent method and the chain rule:

$$\begin{aligned} \tfrac{\partial J(w,b)}{\partial w_i} &= -\tfrac{1}{m} \sum_{j}^{m} [\hat{y}^{(j)}(1 - f_{w,b}(x^{(j)}))x_i^{(j)} - (1 - \hat{y}^{(j)})(1 - f_{w,b}(x^{(j)}))x_i^{(j)}) \\ &= -\tfrac{1}{m} \sum_{j}^{m} [\hat{y}^{(j)} - \hat{y}^{(j)} f_{w,b}(x^{(j)}) - f_{w,b}(x^{(j)}) + \hat{y}^{(j)} f_{w,b}(x^{(j)})]x_i^{(j)} \\ &= -\tfrac{1}{m} \sum_{j}^{m} (\hat{y}^{(j)} - f_{w,b}(x^{(j)}))x_i^{(j)} \end{aligned} \tag{16}$$

According to the gradient descent method, the update iteration expression of w_i is:

$$w_i{}^{(l+1)} = w_i{}^{(l+1)} - lr\frac{\partial J(w,b)}{\partial w_i} = w_i{}^{(l+1)} - \frac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f_{w,b}(x^{(j)}))x_i^{(l+1,j)} \tag{17}$$

Similarly, the update iteration expression of b_i is:

$$b_i{}^{(l+1)} = b_i{}^{(l+1)} - lr\frac{\partial J(w,b)}{\partial b_i} = b_i{}^{(l+1)} - \frac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f_{w,b}(x^{(j)})) \tag{18}$$

where lr represents the learning rate.

The softmax function is often used as a multiclassification activation function in deep learning models, as it is an extension of logistic regression. The features of this function are that the model has multiple outputs and that the number of outputs equals the number of categories. The output value is the probability of the input sample X belonging to each category. Finally, the highest probability denotes that the sample is predicted to belong to the corresponding category. For a learning model has L hidden layers that uses the softmax function as the activation function of the output layer, its mathematical description is as follows:

$$\begin{cases} z_k^{(l+1)} = \vec{w}_k^{(l+1)} \cdot \vec{x}^{(l+1)} + b_k^{(l+1)} \\ f_{w,b}\left(\vec{x}\right) = \frac{e^{z_k^{(l+1)}}}{\sum_{n=1}^{k} e^{z_n^{(l+1)}}} = \frac{e^{\vec{w}_k^{(l+1)} \cdot \vec{x}^{(l+1)} + b_k^{(l+1)}}}{\sum_{n=1}^{k} e^{\vec{w}_n^{(l+1)} \cdot \vec{x}^{(l+1)} + b_n^{(l+1)}}} \end{cases} \tag{19}$$

For a sample with a batch size m, the target is to learn through the model, so the target loss function (also known as the logarithmic likelihood cost function) is:

$$J(w,b) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{k=1}^{K} A\left\{\hat{y}^{(j)} = k\right\}\log\left(\frac{e^{\vec{w}_k^{(l+1,j)} \cdot \vec{x}^{(l+1,j)} + b_k^{(l+1)}}}{\sum_{n=1}^{K} e^{\vec{w}_n^{(l+1,j)} \cdot \vec{x}^{(j)} + b_n^{(l+1)}}}\right) \tag{20}$$

where A$\{\cdot\}$ is the indicator function, which means that when $\hat{y}^{(j)} = k$ is true, the function value is 1; otherwise, the function value is 0. According to binary classification, the logarithmic cost function is transformed into the cross-entropy loss function to obtain the learning objective loss function, as shown in Equation (21):

$$j(w,b) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{k=1}^{k}\left(A\left\{\hat{y}^{(j)} = k\right\}\log(e^{\vec{w}_k^{(l+1)} \cdot \vec{x}^{(l+1,j)} + b_k^{(l+1)}}) - 1\left\{\hat{y}^{(j)} = k\right\}\log\left(\sum_{n=1}^{k} e^{\vec{w}_n^{(l+1)} \cdot \vec{x}^{(l+1,j)} + b_n^{(l+1)}}\right)\right) \tag{21}$$

Equation (21) can be simplified as:

$$J(w,b) = -\frac{1}{m}\sum_{j=1}^{m}\left(\sum_{k=1}^{K}\left(A\left\{\hat{y}^{(j)} = k\right\}\log e^{\vec{w}_k^{(l+1)} \cdot \vec{x}^{(l+1)j)} + b_k^{(l+1)}}\right) - \log\left(\sum_{n=1}^{K} e^{\vec{w}_n^{(l+1)} \cdot \vec{x}^{(l+1,j)} + b_n^{(l+1)}}\right)\right) \tag{22}$$

According to the gradient descent minimization loss function and the chain rule, the partial derivative of $J(w,b)$ is calculated with respect to $\vec{w}_k$.

$$\frac{\partial J(w,b)}{\partial \vec{w}_k^{(l+1)}} = -\frac{1}{m}\sum_{j=1}^{m}\left(\sum_{k=1}^{K}\left(A\left\{\hat{y}^{(j)} = k\right\}\frac{e^{\vec{w}_k^{(l+1)} \cdot \vec{x}^{(l+1,j)} + b_k^{(l+1)}} * \vec{x}^{(l+1,j)}}{e^{\vec{w}_k^{(l+1)} \cdot \vec{x}^{(l+1,j)} + b_k^{(l+1)}}} - \frac{e^{\vec{w}_k^{(l+1)} \cdot \vec{x}^{(l+1,j)} + b_k^{(l+1)}} * \vec{x}^{(l+1,j)}}{\sum_{n=1}^{K} e^{\vec{w}_n^{(l+1)} \cdot \vec{x}^{(l+1,j)} + b_n^{(l+1)}}}\right)\right) \tag{23}$$

Equation (23) is simplified as follows:

$$\begin{aligned}\frac{\partial J(w,b)}{\partial \vec{w}_k^{(l+1)}} &= -\frac{1}{m}\sum_{j=1}^{m}\Big(\sum_{k=1}^{K}\Big(A\Big\{\hat{y}^{(j)}=k\Big\}\vec{x}^{(l+1,j)} - P\big(\hat{y}^{(j)}=k\Big|x^{(l+1,j)}\big)*\vec{x}^{(l+1,j)}\Big)\\ &= -\frac{1}{m}\sum_{j=1}^{m}\Big(\vec{x}^{(l+1,j)} - P\big(\hat{y}^{(j)}=k\Big|x^{(l+1,j)}\big)*\vec{x}^{(l+1,j)}\Big)\end{aligned} \tag{24}$$

According to the gradient descent method, the update iteration expression of $\vec{w}_k$ is:

$$\vec{w}_k^{(l+1)} = \vec{w}_k^{(l+1)} - lr\frac{\partial J(w,b)}{\partial w_k} = \vec{w}_k^{(l+1)} - \frac{lr}{m}\sum_{j=1}^{m}\Big(\vec{x}^{(l+1,j)} - P\big(\hat{y}^{(j)}=k\Big|x^{(l+1,j)}\big)*\vec{x}^{(l+1,j)}\Big) \tag{25}$$

Through the gradient descent method, the iterative expression of b_k can also be obtained as follows:

$$b_k^{(l+1)} = b_k^{(l+1)} - lr\frac{\partial J(w,b)}{\partial b_k} = b_k^{(l+1)} - \frac{lr}{m}\sum_{j=1}^{m}\big(1 - P(y^{(j)}=k|\vec{x}^{(l+1,j)})\big) \tag{26}$$

Through the above analysis, the learning rate and batch size directly determine the updating of the weight and bias of the deep learning model, and there is a correlation between them, as shown in Equations (17), (18), (25), and (26). The learning rate and batch size are important parameters that affect the performance of a deep learning model.

4. Proposed Generalized Mathematical Derivation Considering Multiple Parameters

After the analysis in Sections 3.2–3.4, it is known that the learning rate, dropout rate, batch size, and kernel size all impact the performance of a deep learning model. In particular, the learning rate and batch size directly affect the final loss function parameters of the deep learning model output layer (weight and bias). The number of update iterations and the optimal value ranges of the learning rate, and batch size can be found through the relationship between the two. Therefore, studying the correlations between multiple parameters is of great significance for our choice of hyperparameters.

According to Equations (7) and (8), it is assumed that dropout is adopted in the deep learning model layers, and the loss function parameter updates (Equations (17) and (18)) for the binary classification problem can be expressed as:

$$\begin{aligned}w_i^{(l+1)} &= w_i^{(l+1)} - \tfrac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f_{w,b}(x^{(j)}))x_i^{(l+1,j)}\\ &= w_i^{(l+1)} - \tfrac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f_{w,b}(\sum_i w_i^{(l+1)}\widetilde{y}_i{}^{l} + b_i^{(l+1)}))x_i^{(l+1,j)}\\ &= w_i^{(l+1)} - \tfrac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f(\sum_i w_i^{(l+1)}r^{(l)} * y_i{}^{(l)} + b_i^{(l+1)})x_i^{(l+1,j)}\\ &= w_i^{(l+1)} - \tfrac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f(\sum_i w_i^{(l+1)}r^{(l)} * f(\sum_i w_i^{(l)} * y_i{}^{(l)} + b_i^{(l)}) + b_i^{(l+1)})x_i^{(l+1,j)}\end{aligned} \tag{27}$$

$$\begin{aligned}b_i^{(l+1)} &= b_i^{(l+1)} - \tfrac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f_{w,b}(x^{(j)}))\\ &= b_i^{(l+1)} - \tfrac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f_{w,b}(\sum_i w_i^{(l+1)}\widetilde{y}_i{}^{l} + b_i^{(l+1)}))\\ &= b_i^{(l+1)} - \tfrac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f_{w,b}(\sum_{i=1} w_i^{(l+1)}r^{(l)} * y_i{}^{(l)} + b_i^{(l+1)}))\\ &= b_i^{(l+1)} - \tfrac{lr}{m}\sum_{j}^{m}(\hat{y}^{(j)} - f_{w,b}(\sum_{i=1} w_i^{(l+1)}r^{(l)} * f(\sum_{i=1} w_i^{(l)}y_i{}^{(l)} + b_i^{(l)}) + b_i^{(l+1)}))\end{aligned} \tag{28}$$

In a multiclassification problem, it is assumed that the dropout model is used in layer l of the deep learning model; according to $\frac{e^{\vec{w}_k^{(l+1)} \cdot \vec{x}^{(l+1)} + b_k}}{\sum_{n=1}^{K} e^{\vec{w}_n^{(l+1)} \cdot \vec{x}^{(l+1)} + b_n^{(l+1)}}} = P(\hat{y} = k \Big| \vec{x})$ and Equation (8), the parameter update iteration (Equation (27)) of the loss function (Equation (28)) can be expressed as:

$$\begin{aligned}
\vec{w}_k{}^{(l+1)} &= \vec{w}_k{}^{(l+1,j)} - lr\frac{\partial J(w,b)}{\partial w_k} = \vec{w}_k - \frac{lr}{m}\sum_{j=1}^{m}(\vec{x}^{(j)} - P(y^{(k)} = k \Big| x^{(l+1,j)}) * \vec{x}^{(l+1,j)}) \\
&= \vec{w}_k - \frac{lr}{m}\sum_{j=1}^{m}((\vec{x}^{(j)} - \frac{e^{\vec{w}_k^{(l+1)} \cdot \vec{x} + b_k}}{\sum_{n=1}^{K} e^{\vec{w}_n^{(l+1)} \cdot \vec{x} + b_n^{(l+1)}}}) * \vec{x}^{(l+1,j)}) \\
&= \vec{w}_k - \frac{lr}{m}\sum_{j=1}^{m}((\vec{x}^{(l+1,j)} - f(\sum_i \vec{w}_{k,i}^{(l+1)} \cdot x_i{}^{(l+1,j)} + b_{k,i}^{(l+1)}) * \vec{x}^{(l+1,j)}) \\
&= \vec{w}_k - \frac{lr}{m}\sum_{j=1}^{m}((\vec{x}^{(l+1,j)} - f(\sum_i \vec{w}_{k,i}^{(l+1)} r^{(l)} * y_i{}^{(l,j)} + b_{k,i}^{(l+1)}) * \vec{x}^{(l+1,j)}) \\
&= \vec{w}_k - \frac{lr}{m}\sum_{j=1}^{m}((\vec{x}_i{}^{(l+1,j)} - f(\sum_{i=1} w_i^{(l+1)} r^{(l)} * f(\sum_{i=1} w_i^{(l)} y_i{}^{(l)} + b_i^{(l)}) + b_{k,i}^{(l+1)}) * \vec{x}^{(l+1,j)})
\end{aligned} \tag{29}$$

$$\begin{aligned}
b^{l+1}{}_k &= b^{(l+1)}{}_k - lr\frac{\partial Jw,b}{\partial w_k} = b_k - \frac{lr}{m}\sum_{j=1}^{m}(\vec{x}^{(j)} - P(y^{(k)} = k \Big| x^{(l+1,j)})) \\
&= b^{(l+1)}{}_k - \frac{lr}{m}\sum_{j=1}^{m}((\vec{x}^{(j)} - \frac{e^{\vec{w}_k^{(l+1)} \cdot \vec{x} + b_k^{(l+1)}}}{\sum_{n=1}^{K} e^{\vec{w}_n^{(l+1)} \cdot \vec{x} + b_n^{(l+1)}}})) \\
&= b^{(l+1)}{}_k - \frac{lr}{m}\sum_{j=1}^{m}((1 - f(\sum_i \vec{w}_{k,i}^{(l+1)} \cdot x_i{}^{(l+1,j)} + b_{k,i}^{(l+1)})) \\
&= b^{(l+1)}{}_k - \frac{lr}{m}\sum_{j=1}^{m}((1 - f(\sum_i \vec{w}_{k,i}^{(l+1)} r^{(l)} * y_i{}^{(l,j)} + b_{k,i}^{(l+1)})) \\
&= b^{(l+1)}{}_k - \frac{lr}{m}\sum_{j=1}^{m}((\vec{x}_i{}^{(l+1,j)} - f(\sum_{i=1} w_{k,i}^{(l+1)} r^{(l)} * f(\sum_{i=1} w_{k,i}{}^{(l)} y_i{}^{(l)} + b_i^{(l)}) + b_{k,i}^{(l+1)}))
\end{aligned} \tag{30}$$

According to the above analysis, the dropout rate is also an important parameter affecting the performance of the deep learning model, and it has certain correlations with the learning rate and batch size, as shown in Equations (27)–(30). The range value for parameter optimization can be found through these correlation relationships.

Moreover, when we calculate the number of parameters used by the deep learning model according to the size of the convolution kernel, the number of parameters is also related to whether dropout is used.

If the convolution layer adopts dropout, according to Equation (4) and the dropout principle, the number of parameters involved in the calculation of the convolution layer is:

$$\begin{cases} w_c = (1-q)ke^2 \times ch \times N \\ B_c = N \\ P_c = (1-q)W_c - B_c \end{cases} \tag{31}$$

If the fully connected layer adopts dropout, the number of parameters of the fully connected layer connected to the convolutional layer according to Equation (5), and the dropout principle is calculated as shown in Equation (32):

$$\begin{cases} w_{cf} = (1-q)ke^2 \times ch \times N \\ B_{cf} = N \\ P_{cf} = (1-q)W_{cf} - B_{cf} \end{cases} \tag{32}$$

According to Equation (6) and the dropout principle, the calculation processes for all parameters connected to the fully connected layer are shown in Equation (33):

$$\begin{cases} W_{ff} = (1-q)F_{-1} \times F \\ B_{ff} = F \\ P_{ff} = (1-q)W_{ff} - B_{ff} \end{cases} \tag{33}$$

Through Equations (32) and (33), we could calculate the number of parameters involved in the calculation of each layer of the deep neural network model when we used dropout, which helped us effectively understand the deep learning model.

5. Experiments and Analyses

After the correlation analysis in Section 4 we know that the four hyperparameters (the kernel size, learning rate, dropout rate and batch size) theoretically have an impact on the performance of the deep learning model. To better study the influence of each hyperparameter on model performance, in this section, we designed experiments according to the single-factor experimental method. In this section, we conducted hyperparameter experiments on the MNIST dataset [54]. The MNIST dataset is a publicly available dataset that has been adopted by many studies in image processing and has almost become a "paradigm". The MNIST training data set contains 60,000 samples and the test data set contains 10,000 samples, and each image in the MNIST dataset consists of 28 × 28 pixel points, each represented by a grayscale value. To avoid randomness, the training accuracy was defined as the average value calculated after we repeated each parameter experiment three times. In these experiments, the adopted multilayer neural network architecture consisted of three convolution layers and three fully connected layers. In addition to using softmax as the classification function for the output layer, the activation function of the hidden layer was selected as a leaky rectified linear unit "Leaky-ReLU (LReLU)" [55]. The reason for choosing this network was that it is easy to train and sufficient for carrying out experiments on the performance of the hyperparameter-influenced learning model. In order to ensure the accuracy of the experimental results, 10 cross validation tests were carried out on LRelu, Relu [50], and scaled exponential linear unit (Selu) [51] models, respectively.

5.1. Experimental Environment

The Python programming language was used for coding a CNN learning algorithm, which was implemented on a 64-bit Ubuntu 18.04.5 LTS. TensorFlow 1.14.0 was installed to evaluate the performance of the designed model. The model performance was tested and analyzed under different q, m, ke, and lr, and the relative optimal parameters were obtained. Then, the optimal parameters were combined and verified. All results were tested on the same computer model, and the main configuration of the computer was as follows: an Intel (R) Core (TM) i7-9700F CPU @ 3.00 GHz, 8.00 GB of RAM, and an NVIDIA GeForce GTX 1660 GPU.

5.2. Experimental Design

We conducted two experiments: an experiment on the influence of a single parameter on the performance of the deep learning model, and an optimization combination verification experiment. The purpose of the former was to verify the influences of the q, m, lr, and ke on model performance, and the optimal value range of each parameter was obtained through experiments. On the basis of the former, the purpose of the latter was to combine the optimal range values of all parameters obtained from the former to further verify the influence of each parameter on the performance of the deep learning model.

According to the first purpose, we designed experiments as presented in Section 5.4. The variables of our experiments were four hyperparameters: the dropout rate, batch size, learning rate, and kernel size. Each hyperparameter variable was valued at equal intervals, and the number of training steps was 100,000. According to the second purpose, we designed an experiment as shown in Section 5.5 with 100,000 training steps.

5.3. Hyperparameter Selection

Equations (27)–(33) prove that the hyperparameters (q, m, ke, lr) have an effect on the weight and bias of the model when other parameters are constant. They show that m and lr are important parameters affecting the updating of model weights. If lr is too large, the model does not converge; if lr is too small, the model converges very slowly or cannot be learned, and the size of m value affects the model convergence to a certain extent [52,53,56]. The value of m is often a power of 2 [57]. Equations (7)–(9) indicate that the selection of the q value is one of the keys to parameter tuning, and directly affects the robustness of the model. According to the principle of dropout and the rules of neural network parameter updating, when m, lr, and ke are constant values, the value of q ranges from {0.3,0.5} [43,45,56]. The ke value directly determines the parameters and computation required by the model. If the value is too small, the model's receptive field is not enough to learn features; if the value is too large, the model's parameters are too many to lead to computation, the ke value range is {3, 7} [49].

5.4. Experiment and Analysis Based on Single Parameter Optimization

(A) On the q

To study the effect of the dropout rate on the examined deep learning model, we set different settings as q = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}, and the other three parameters were m = 64, ke = 5, and $lr = 1 \times 10^{-4}$.

According to Figure 1, the training accuracy of the network model varied with different dropout rates as follows: when q = {0.1, 0.2, 0.3, 0.7}, the training accuracy was 100%; when q = {0.4, 0.5, 0.6, 0.8, 0.9}, the training accuracies were 99.48%, 99.48%, 98.44%, 95.83%, and 89.06%, respectively. It can be concluded that the average accuracy did not change when the dropout rate was between 0.1–0.3 in the training stage. Then, when q increased from 0.4 to 0.7, the accuracy range was very small, but when q increased from 0.7 to 0.9, the training accuracy decreased sharply. The convergence of the accuracy rate is shown in Figure 2. As shown in Figure 2a, the fastest convergence rate on the training set was achieved when q = {0.5, 0.4, 0.3}. Figure 2b shows that the convergence rate was relatively fast for the test set when q = {0.5, 0.4, 0.3}.

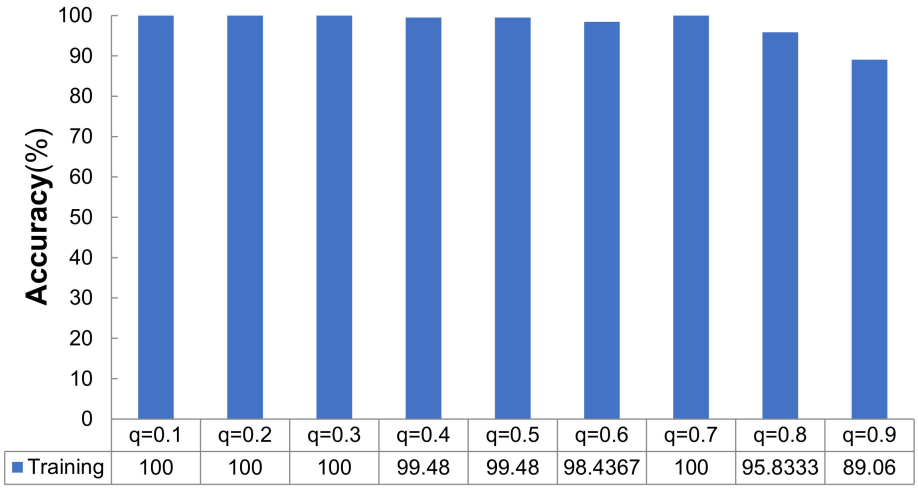

Figure 1. Training accuracies obtained under different q (%).

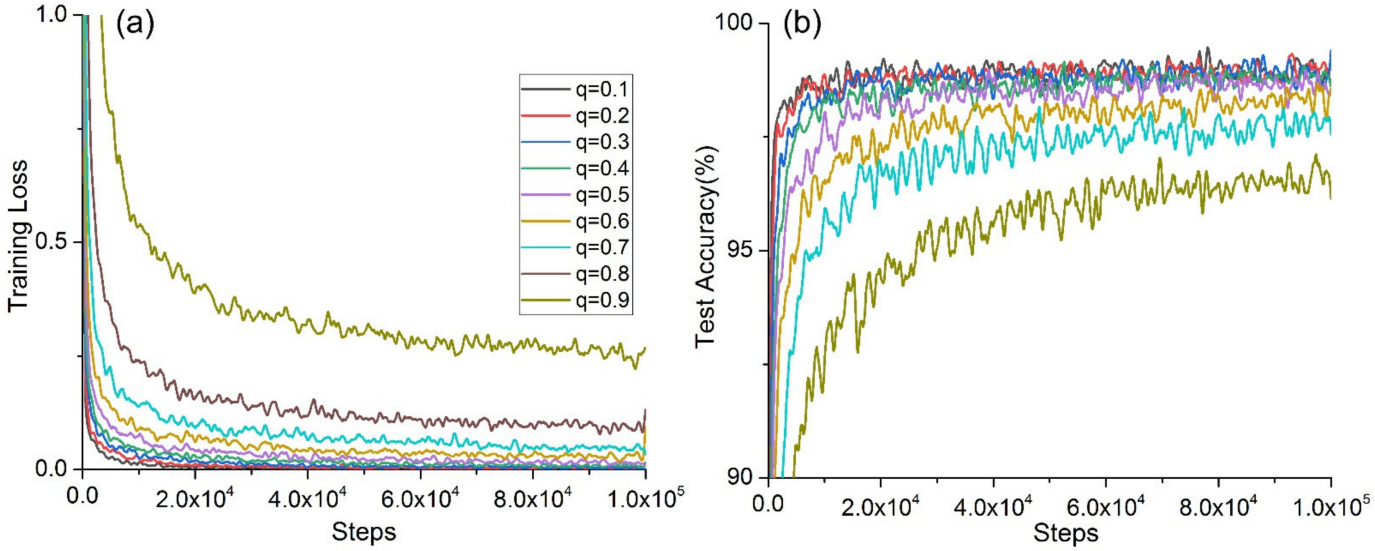

Figure 2. Model convergence under different q: (**a**) convergence of training cross-entropy loss; (**b**) convergence of test accuracy.

Table 1 shows that the variation in time consumption could be divided into three distinct stages: in the first stage, when $q < 0.4$, the time consumption increased with the increase in the dropout value; in the second stage, when $0.4 \leq q \leq 0.6$, the time taken for $q = 0.4$ decreased relative to that required for $q = 0.3$, but when the dropout increases from 0.4 to 0.6, the time increased with the increase in the dropout value. Stage 3: when $q = 0.6$ increased to $q = 0.8$, the time decreased as the dropout value increased. However, when $q = 0.8$ and $q = 0.9$, there was no significant change in the time consumption of the operation.

Table 1. The running time required with the same number of steps for different q.

q	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Running time (s)	10,094	10,118	10,270	10,023	10,089	10,240	10,192	10,041	10,060

Considering the convergence of the accuracy and cross-entropy loss values during model training, the convergence of the accuracy during testing and the time consumption incurred when going through the same steps, the experiment showed that the best performance obtained by the deep neural network model using dropout occurred when the dropout rate was set as $q = \{0.3, 0.4, 0.5\}$.

(B) On the m

The m is an important parameter in machine learning that is set in a reasonable range; it can improve the memory utilization rate of the computer, improve its processing speed with the same amount of data, and reduce the training shock caused by the decline in memory. To observe the changes in the performance of the multilayer CNN model when the batch size was set to different values, we designed an experiment in this section. The parameter variable of this experiment was the batch size value; i.e., the other parameters were held constant: $q = 0.5$, $lr = 1 \times 10^{-4}$, and $ke = 5$.

Figure 3 shows that with increasing m, the average accuracy rate did not change and reached 100%. Figure 4a shows the change in the cross-entropy loss with the same number of steps for the network model on the training set at different m values, and Figure 4b shows the accuracy of the network model on the test set with different m. Figure 4a shows that when $m = 2$, the convergence was slowest and exhibited strong oscillation; when $m = 4$, the convergence and oscillation were second only to those in the case when $m = 2$; when $m = 64$ and $m = 32$, the convergence was fast and the oscillation was small. Figure 4b shows that when $m = 2$, the convergence was slowest, and exhibited strong oscillation; when

m = 4, its convergence and oscillation were second only to those in the case when m = 2; when m = 64 and m = 32, its convergence was fast, and the oscillation was small.

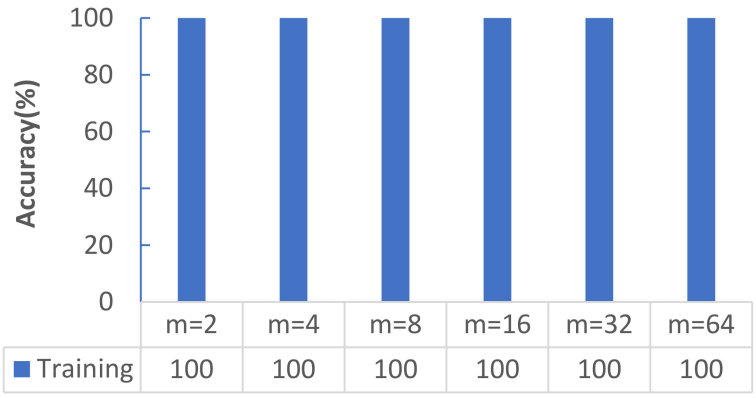

Figure 3. Training accuracies obtained under different m (%).

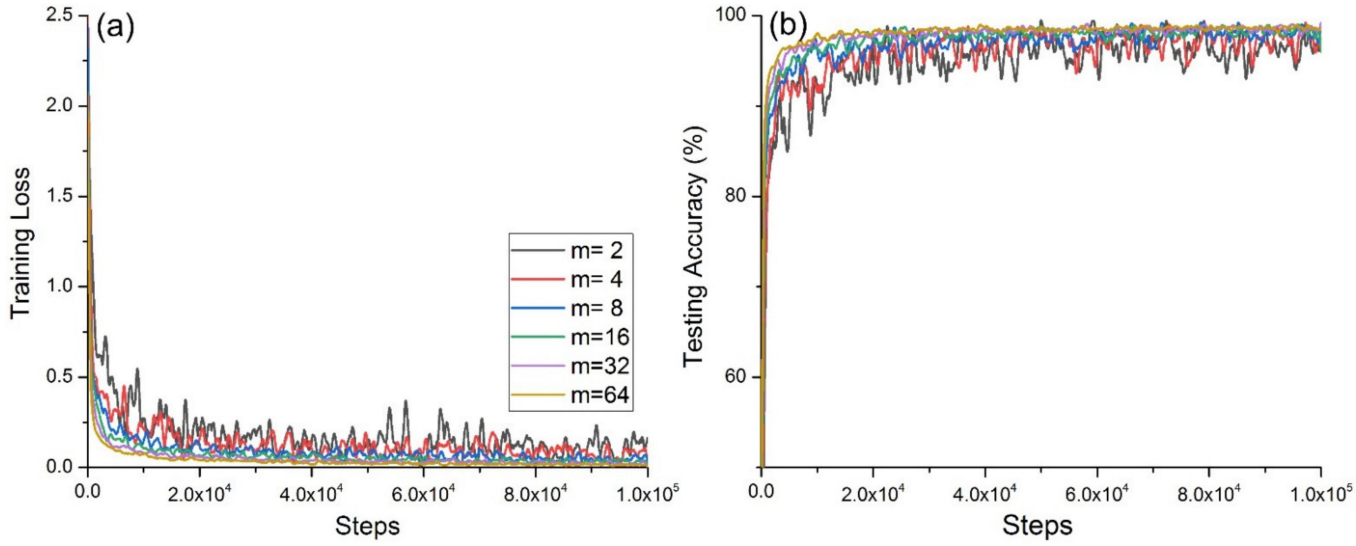

Figure 4. Model convergence under different m: (**a**) convergence of training cross-entropy loss; (**b**) convergence of test accuracy.

When m = {2,4,8,16}, the accuracy was high, but the training cross-entropy loss value and test accuracy oscillated greatly, and the convergence was poor. The reason for this was that the number of samples fed each time was too small, which led to overfitting during the model training process.

Table 2 shows that as m continued to increase, the running time for model also increased.

Table 2. The running time required with the same number of steps at different m.

m	2	4	8	16	32	64
Running time (s)	1482	2101	2352	3474	5677	10,089

The above experimental results were in line with Equations (25) and (26). The m is one of the main hyperparameters that affects the learning performance of the deep model, and directly affects the parameter update of the cross-entropy loss function of the deep learning model.

Considering the convergence of the accuracy and cross-entropy loss during model training, the convergence of the accuracy during testing, and the running time incurred during the same steps, the experiment proved that when m = 32 was used, the performance of the deep neural network model reached its maximum.

(C) On the *lr*

The *lr* has a key impact on the performance of the model. Too large an *lr* causes the model to fail to converge, while a small rate causes the model to converge very slowly or fail to learn. To study the effects of different *lr* on the performance of a multilayer CNN, the parameter variables were $lr = \{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}, 1 \times 10^{-7}\}$, while the other three parameters were held constant: $q = 0.5$, $m = 32$, and $ke = 5$.

It can be seen from Figure 5 that when the learning rate was in the range $\{1 \times 10^{-2}, 1 \times 10^{-3}\}$, the average accuracy rate increased with a decreasing learning rate; and when $lr = 1 \times 10^{-4}$ and *lr* = 1e-5, the average accuracy rate reached 100%. When $lr < 1 \times 10^{-5}$, the average accuracy dropped sharply, first to 90.6267%, and finally to 58.3367%.

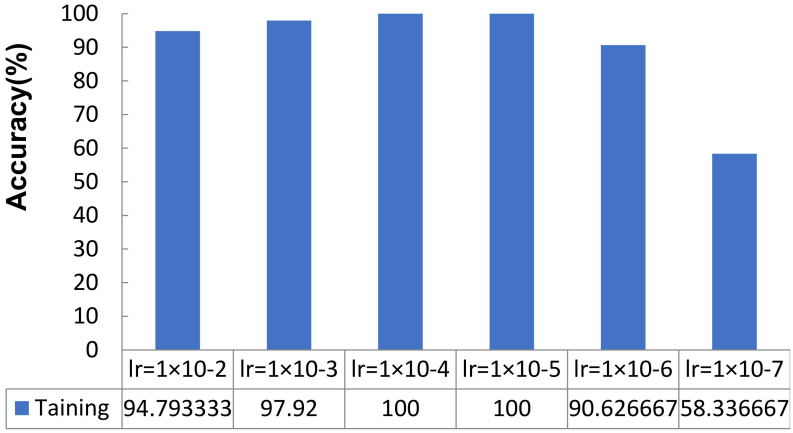

Figure 5. Training accuracies obtained under different *lr* (%).

Figure 6 shows that when $lr = 1 \times 10^{-7}$, both the training loss and testing accuracy failed to converge due to the low learning rate. When $lr = 1 \times 10^{-6}$, the convergence rate of the model was slow, and was only faster than that obtained when $lr = 1 \times 10^{-7}$. This experimental result is described in Equations (25) and (27); the learning rate is one of the main hyperparameters that affects the learning performance of the deep model and directly affects the parameter update of the cross-entropy loss function of the deep learning model. Figure 6b shows that the convergence rate of the deep CNN increased with an increasing learning rate. However, when $lr = 1 \times 10^{-2}$, its convergence worsened, and its oscillation increased. When going through the same steps, the time required by the model differed little, as shown in Table 3.

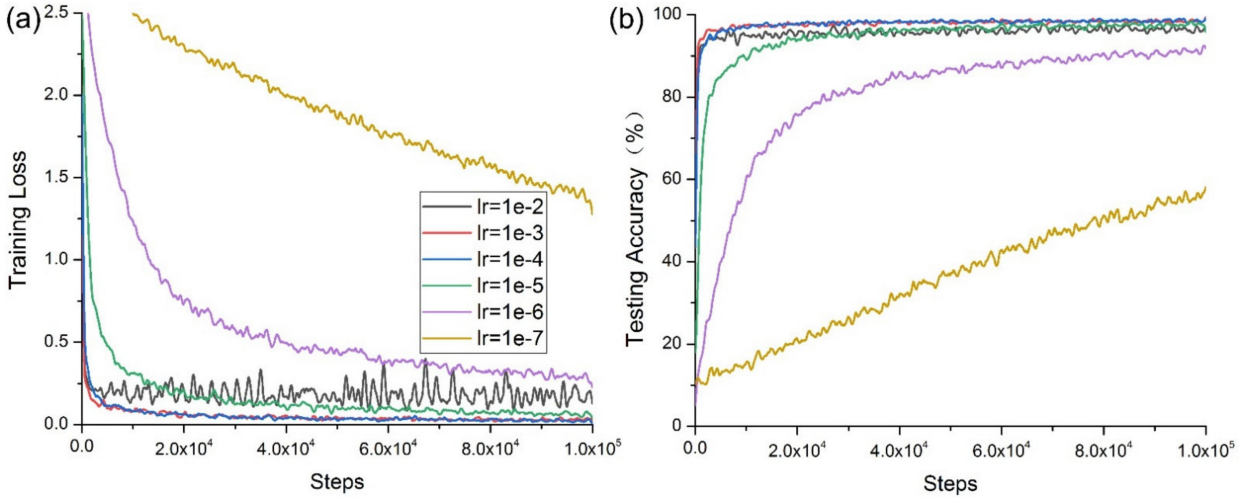

Figure 6. Model convergence with different *lr*: (**a**) convergence of training cross-entropy loss; (**b**) convergence of testing accuracy.

Table 3. The running time required with the same number of steps at different *lr*.

lr	1×10^{-2}	1×10^{-3}	1×10^{-4}	1×10^{-5}	1×10^{-6}	1×10^{-7}
Running time (s)	5596	5619	5677	5496	5554	5514

Experiments proved that the depth of the initial vector on CNN models had an impact on the performance, and when $lr = 1 \times 10^{-3}$ and $lr = 1 \times 10^{-4}$, the depth of the neural network model attained the best performance condition.

(D) On the *ke*

The convolution process multiplies the elements in the convolution kernel and the corresponding pixels in the image successively, and sums them as the new pixel value after convolution. Then, the convolution kernel is shifted along the original image, and new pixel values are calculated until the whole image is covered. This process can produce many different effects according to different image convolution results.

In this section, experiments were designed to test the effects of different kernel sizes on the performance of the multilayer CNN. This set of experimental kernel sizes was used as a variable parameter, and its value was *ke* = {1,2,3,4,5,6,7}. The other three parameters were held constant: $q = 0.5$, $m = 32$, and $lr = 1 \times 10^{-4}$.

When the other parameters of the network model remained unchanged, the accuracy value of the model changed in two stages with increasing kernel size: *ke* = {1,2,3,4} (stage 1) and *ke* = {5,6,7} (stage 2). The change trends in the two stages showed that the accuracy rate of the model increased with increasing kernel size, as shown in Figure 7.

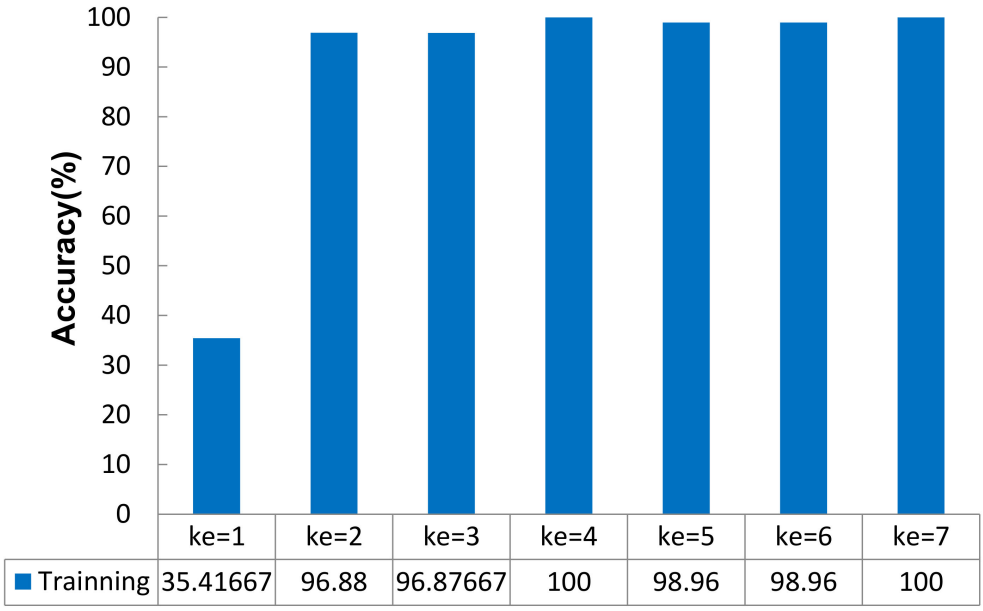

Figure 7. Training accuracies obtained under different *ke* (%).

In Figure 8a, when *ke* = 1, the convergence was slowest and exhibited strong oscillation. When *ke* = 2, the oscillation was second only to those obtained when *ke* = 1. When *ke* > 3, its convergence was good, and there were no obvious changes in the kernel size. In Figure 8b, when *ke* = 1, the convergence was slowest and exhibited strong oscillation. When *ke* = 2, the convergence and oscillation were second only to those obtained when *ke* = 1. When *ke* > 3, the convergence was good, and there were no obvious changes in the *ke* value.

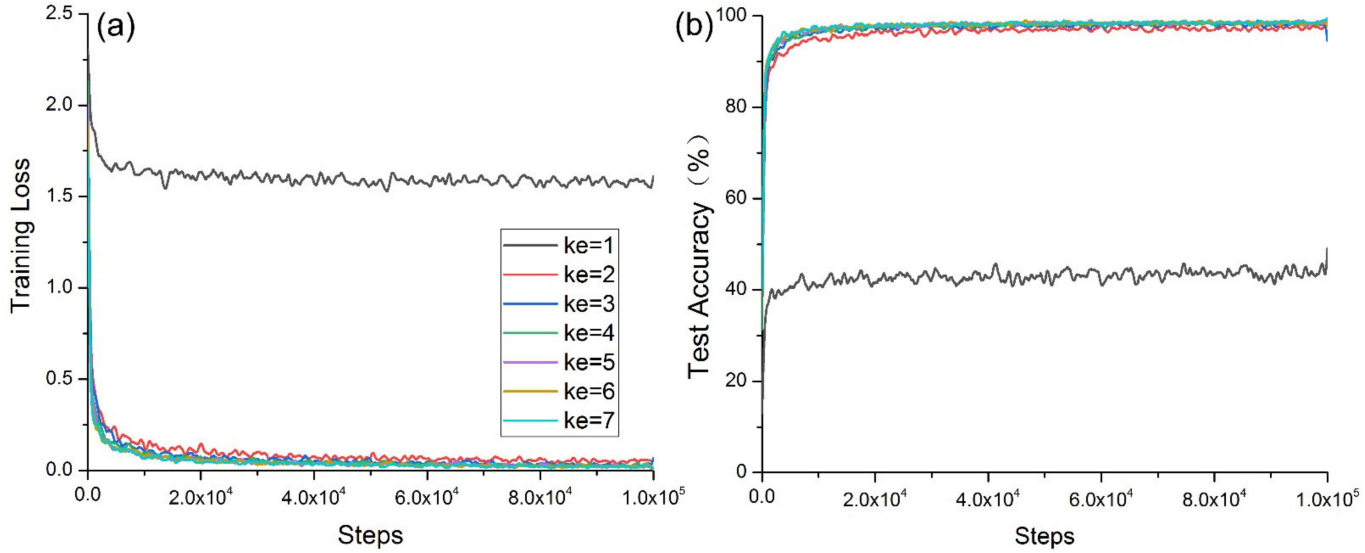

Figure 8. Model convergence with *ke*: (**a**) convergence of training cross-entropy loss; (**b**) convergence of test accuracy.

With an increasing *ke* value, the time consumed by the model for learning could be divided into two stages, as shown in Table 4. The time consumption at *ke* = 2 was 654 s lower than that at *ke* = 1. The time consumption at *ke* = 3 was 621 s lower than that at *ke* = 2. The time consumption at *ke* = 4 was 143 s higher than that at *ke* = 3. The time consumption at *ke* = 5 was 241 s higher than that at *ke* = 6. The time consumption at *ke* = 7 was 151 s higher than that at *ke* = 6.

Table 4. The running time required for the same number of steps at different *ke*.

ke	1	2	3	4	5	6	7
Running time (s)	6719	6064	5443	5586	5677	5918	6069

According to the four indicators of model training accuracy, cross-entropy loss value convergence, test accuracy convergence, and the time spent during the same step, the experiments showed that *ke* had an impact on the performance of the deep CNN model. The optimal value range of the kernel size was *ke* = {4, 5}, and when *ke* = 4, the performance of the deep CNN model reached the best state.

5.5. Confirmatory Experiments and Analyses on Multiparameter Optimization

According to the experimental results obtained in Section 5.4 the *q*, *m*, *lr*, and *ke*, which enabled the model to achieve relatively optimal performances, were selected for combined experiments, and their network parameters are shown in Table 5.

Table 5. Network parameter design for the confirmatory experiment.

Experiment Number	*q*	*m*	*ke*	*lr*
1	0.3	32	4	1×10^{-3}
2	0.4	32	4	1×10^{-3}
3	0.5	32	4	1×10^{-3}
4	0.3	32	4	1×10^{-4}
5	0.4	32	4	1×10^{-4}
6	0.5	32	4	1×10^{-4}

Combining the relatively good dropout rate, batch size, learning rate, and convolution kernel size obtained in the experiments in Section 5.4, it was found that the parameter performance of the model was relatively stable at this time. The training accuracy and test accuracy were very similar (almost equal), as shown in Figure 9. However, in terms of convergence (as shown in Figure 10), the first group, the fourth group, and the second group had the best convergence rates, and the sixth group had the weakest convergence. The running time through the same steps appeared in the fifth group the least, followed by the first group and the fourth group. The differences between the time consumptions of the latter group and the time consumptions of the previous group were 29 s, 11 s, −34 s, −17 s, and 35 s (as shown in Table 6). With an improvement in computer performance, especially the performance of the GPU, if the time consumption difference yielded by the deep learning network model with the same number of steps was not large, the main performance indicators could be judged in terms of accuracy and convergence. Therefore, these verification experiments showed that the model had the best performance when the parameters of the model were set to the first group of parameters, namely, $\{q, m, ke, lr\} = \{0.3, 32, 4, 1 \times 10^{-3}\}$. The performance is best when the fourth group is used, that is, $\{q, m, ke, lr\} = \{0.3, 32, 4, 1 \times 10^{-4}\}$. The experimental results verified Equations (29) and (30). When the convolution kernel size and batch size were unchanged, the learning rate and discarding rate were the main parameters that affected the performance of the deep learning model, among which the learning rate had the greatest influence.

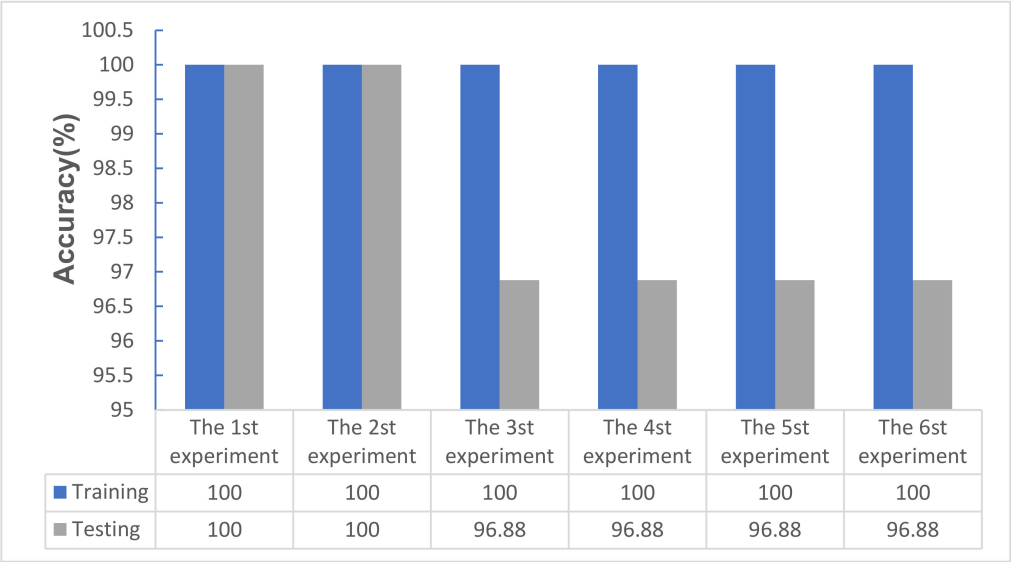

Figure 9. Training accuracies and test accuracies obtained for the confirmatory experiment (%).

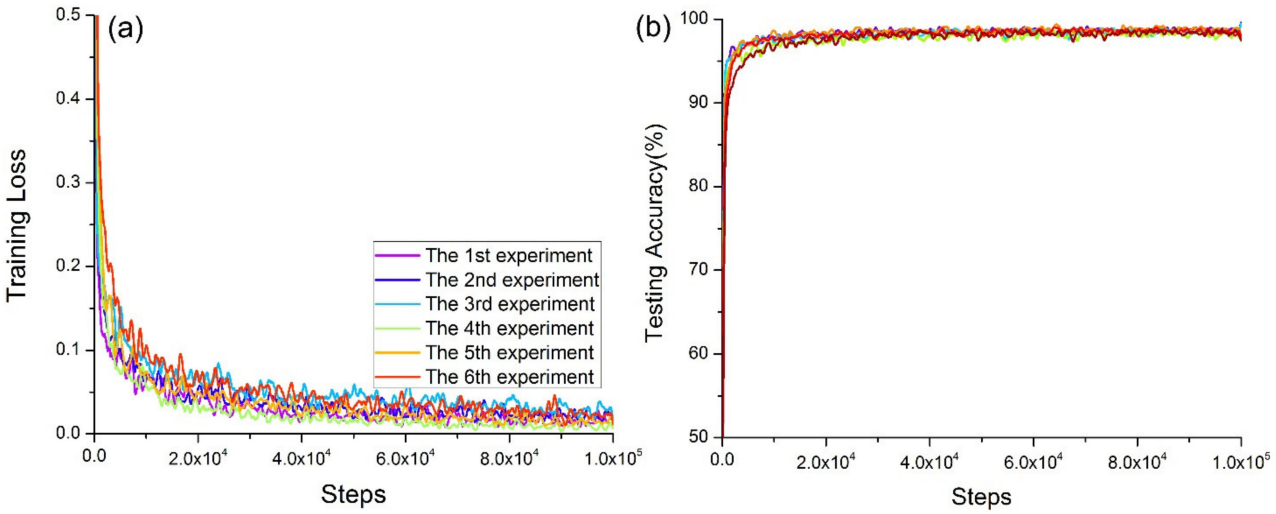

Figure 10. Model convergence for the confirmatory experiment: (**a**) convergence of training cross-entropy loss; (**b**) convergence of test accuracy.

Table 6. The running time required for the same number of steps.

Experiment Number	1st Experiment	2nd Experiment	3rd Experiment	4th Experiment	5th Experiment	6th Experiment
Running time (s)	5298	5337	5358	5314	5297	5332

5.6. Repeatability Validation on Different Architectures

In order to avoid randomness of test results, we performed 10 repetitive cross-validation of the multiparameter experiment under the optimal hyperparameters ($\{q, m, ke, lr\} = \{0.3, 32, 4, 1 \times 10^{-4}\}$). It was guaranteed that they used the same training set for each case and for each variation of the hyperparameter studied. For each of these repetitions, the box diagram was statistically drawn for the results of every 10,000 steps, as shown in Figure 11. In particular, the {ST1 ST2, ST3, ST4, ST5 and ST6, ST7, ST8, ST9, ST10} corresponding steps were {10,000, 20,000, 30,000, 40,000, 50,000, 60,000, 70,000, 80,000, 90,000, 100,000}. Meanwhile, we selected two other activation functions, Relu and Selu, as different architectures to carry out experiments, which helped to validate and achieve stronger conclusions. It can be seen from Figure 11a,b that both the training loss function and the test accuracy rate could reach stability and convergence in the training process when the number of steps was higher than 900,000, which better verified the reliability of our experimental conclusions. To verify and draw convincing conclusions, we tested the above experimental values $\{q, m, ke, lr\} = \{0.3, 32, 4, 1 \times 10^{-4}\}$ on the activation functions Relu and Selu, respectively, as shown in Figure 11c–f. It can be concluded that both the test accuracy rate and training loss function of the Relu and Selu model could show stability and convergence in the training process, which verified the reliability of our experimental conclusions. In other words, it indicated that the experimental results were plausible and were consistent with the conclusions of the mathematical derivation given in Section 4.

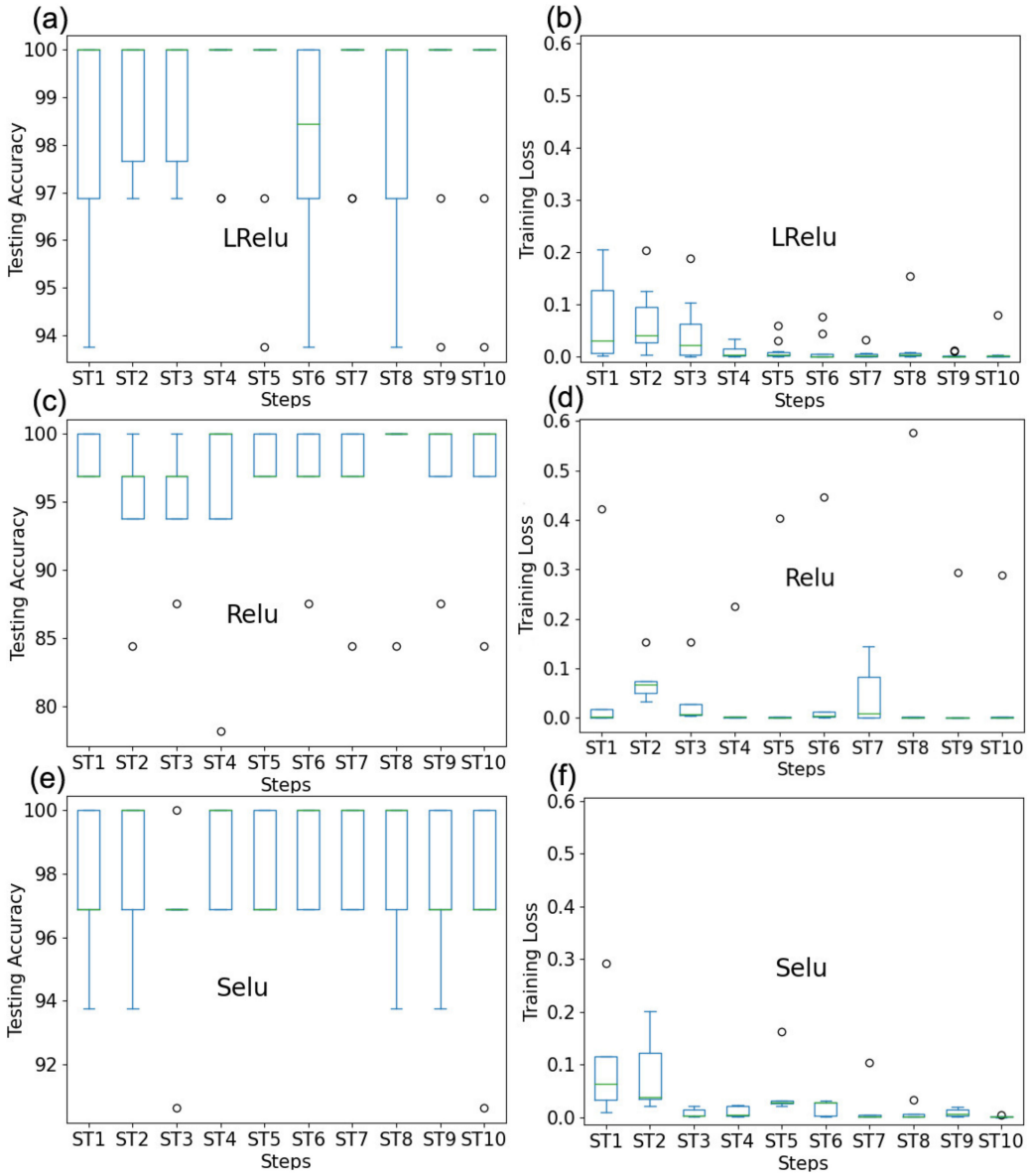

Figure 11. Statistical box plots of LRelu, Relu, and Selu models under different steps for 10 times experiment results: (**a**) testing accuracy of LRelu; (**b**) training loss of LRelu; (**c**) testing accuracy of Relu; (**d**) training loss of Relu; (**e**) testing accuracy of Selu; (**f**) training loss of Selu.

6. Conclusions

Although empirical and adaptive parameter-optimization models are often used to improve the performances of CNN models, empirical parameter models' narrow limitations and adaptive parameter optimization models' insufficient convergence rates or precondition constraints affect their applications and performances. This work attempted to provide analytical expressions for the mathematical relationships between the *q*, *m*, *lr*, and *ke* parameters and reveal the impacts of these four hyperparameters on the performance of a deep learning model in principle. It proposed a multiparameter optimization model for obtaining the optimal hyperparameters; the effectiveness of the proposed model was verified through various experiments based on the MNIST dataset, and the following conclusions could be drawn:

1. Firstly, from the perspective of single parameter, the interpretability of CNN was analyzed from two aspects of mathematical derivation and experimental verification. On this basis, an improved generalized optimization model considering multiple parameters

was proposed to explain the CNN model and guide its parameter optimization, and it was verified by experiments. The mathematical equation that reflected the relationships between the q, m, ke, and lr hyperparameters could be concluded from the mathematical derivation and experimental verification: a. the learning performance was most sensitive to the lr, and cases in which the model did not converge, the convergence was too slow, or the model could not learn attributes were due to a learning rate that was too large or small; b. the ke had a certain impact on the performance of the model, and determined how efficiently the model learned features (the size of the output image) at each layer; and c. although the m and q were not as sensitive as the lr, they were also key parameters that affected model performance.

2. Twenty-eight sets of experiments were carried out by changing a single parameter each time, and we chose q = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}, m = {2, 4, 8, 16, 32, 64}, and lr = $\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}, 1 \times 10^{-7}\}$. Experimental results showed that the best learning model performance could be obtained when the dropout rate, batch size, learning rate, and convolution kernel size were q= {0.3, 0.5}, m = {32, 64}, lr = $\{1 \times 10^{-4}, 1 \times 10^{-4}\}$, and ke = {4,5}, respectively. In particular, when the q = 0.4, m = 32, ke = 4, and lr = 1×10^{-4}, the performance of the model was best.

3. The multi-hyperparameter mathematical correlation model proposed in this paper guided us to effectively perform the hyperparameter tuning process, which could also help us better understand deep learning models. Verification experiments showed that when the hyperparameters were set within reasonable ranges, the performance of the deep learning model was relatively stable, accurate, and convergent, so it could achieve relatively good results.

Author Contributions: Conceptualization, M.S., J.Y. and S.L.; data curation, M.S. and Q.B.; formal analysis, A.Z.; funding acquisition, S.L.; investigation, M.S. and J.Y.; methodology, M.S., J.Y. and S.L. project administration, S.L. and A.Z.; resources, S.L., A.Z. and J.Y.; software, M.S. and Q.B.; supervision, S.L.; validation, M.S. and J.Y.; visualization, M.S.; writing—original draft preparation, M.S.; writing—review and editing, M.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB1713300; in part by the National Key Technologies Research and Development Program of China under Grant 2018AAA0101803; in part by the Guizhou Provincial Colleges and Universities Integrated Tackling Platform Project, UAV Test Flight Integrated Public Relations Platform, Qianjiaohe KY (2020) 005; and in part by the Guizhou Province Major Science and Technology Special Project, Research and Application of Key Technologies for Intelligent Manufacturing of Lithium-Ion Battery Cathode Materials, Qiankehe Major Special Project (2019)3003.

Data Availability Statement: The MNIST datasets are available at http://yann.lecun.com/exdb/mnist/ (accessed on 22 February 2021).

Conflicts of Interest: The authors declare that they have no conflict of interest.

Abbreviations

Symbol	Description
q	Dropout rate
lr	Batch size
m	Learning rate
ke	Convolution kernel size (kernel height = kernel width)
K	Number of sample categories
X	Sample dataset
$z^{(l)}$	Vector input to the l layer
$y^{(l)}$	Output vector from the l layer

$f(z)$	Activation function
$x_i^{(l)}$	Input of the l layer
$w_i^{(l)}$	Weight of the l layer of $x_i^{(l)}$
$b_i^{(l)}$	Bias of the l layer of $x_i^{(l)}$
$\vec{x}_{i,k}^{(l)}$	Input of the sample of the K category in the l layer
$w_{k,i}^{(l)}$	Weight of the l layer of class k
$b_{k,i}^{(l)}$	Bias of the l layer of class k
I	Input image size
S	Stride size
P	Padding size
O	Output image size
W_c	Number of weights the convolutional layer
B_c	Number of biases of the convolutional layer
P_c	Number of all parameters of the convolutional layer
N	Set number of cores
CH	Number of input image channels
W_{cf}	Number of biases of the fully connected layer connected to the convolutional layer
B_{cf}	Number of weights of the fully connected layer connected to the convolutional layer
o'	Output image size of the previous convolution layer
P_{cf}	Total number of parameters in the current fully connected layer
F	Number of neurons in the current fully connected layer
F_{-1}	Number of neurons in the previous fully connected layer

References

1. Lu, S.; Ren, C.; Zhang, J.; Zhai, Q.; Liu, W. A Novel Approach to Droplet's 3D Shape Recovery Based on Mask R-CNN and Improved Lambert-Phong Model. *Micromachines* **2018**, *9*, 462. [CrossRef] [PubMed]
2. Li, C.; Qiu, Z.; Cao, X.; Chen, Z.; Gao, H.; Hua, Z. Hybrid Dilated Convolution with Multi-scale Residual Fusion Network for Hyperspectral Image Classification. *Micromachines* **2021**, *12*, 545. [CrossRef]
3. Alameh, M.; Abbass, Y.; Ibrahim, A.; Valle, M. Smart Tactile Sensing Systems Based on Embedded CNN Implementations. *Micromachines* **2020**, *11*, 103. [CrossRef] [PubMed]
4. Hinton, G.E. A Practical Guide to Training Restricted Boltzmann Machines. In *Neural Networks: Tricks of the Trade*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7700, pp. 599–619.
5. Larochelle, H.; Erhan, D.; Courville, A.; Bergstra, J.; Bengio, Y. An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation. *ACM Int. Conf. Proc. Ser.* **2007**, *227*, 473–480.
6. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* **1998**, *86*, 2278–2323. [CrossRef]
7. Bergstra, J.; Bengio, Y. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
8. Wang, J.; Xu, J.; Wang, X. Combination of Hyperband and Bayesian Optimization for Hyperparameter Optimization in Deep Learning. *arXiv* **2018**, arXiv:1801.01596.
9. Snoek, J.; Larochelle, H.; Adams, R.P. Practical Bayesian Optimization of Machine Learning Algorithms. *Adv. Neural Inf. Process. Syst.* **2012**, *4*, 2951–2959.
10. Eggensperger, K.; Feurer, M.; Hutter, F.; Bergstra, J.; Snoek, J.; Hoos, H.H.; Leyton-Brown, K. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters. In Proceedings of the NIPS 2013 Workshop on Bayesian Optimization in Theory and Practice, Lake Tahoe, NV, USA, 10 December 2013; pp. 1–5.
11. Martinez-de-Pison, F.J.; Gonzalez-Sendino, R.; Aldama, A.; Ferreiro-Cabello, J.; Fraile-Garcia, E. Hybrid Methodology Based on Bayesian Optimization and GA-PARSIMONY to Search for Parsimony Models by Combining Hyperparameter Optimization and Feature Selection. *Neurocomputing* **2019**, *354*, 20–26. [CrossRef]
12. Wang, Y.; Zhang, H.; Zhang, G. CPSO-CNN: An Efficient PSO-Based Algorithm for Fine-Tuning Hyper-Parameters of Convolutional Neural Networks. *Swarm Evol. Comput.* **2019**, *49*, 114–123. [CrossRef]
13. Darwish, A.; Ezzat, D.; Hassanien, A.E. An Optimized Model Based on Convolutional Neural Networks and Orthogonal Learning Particle Swarm Optimization Algorithm for Plant Diseases Diagnosis. *Swarm Evol. Comput.* **2020**, *52*, 100616. [CrossRef]
14. Runge, F.; Stoll, D.; Falkner, S.; Hutter, F. Learning to Design RNA. In Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019.
15. Falkner, S.; Klein, A.; Hutter, F. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, 10–15 July 2018; Volume 4, pp. 2323–2341.
16. Zhang, B.; Rajan, R.; Pineda, L.; Lambert, N.; Biedenkapp, A.; Chua, K.; Hutter, F.; Calandra, R. On the Importance of Hyperparameter Optimization for Model-Based Reinforcement Learning. *PMLR* **2021**, *130*, 4015–4023.

17. Paul, S.; Kurin, V.; Whiteson, S. Fast Efficient Hyperparameter Tuning for Policy Gradient Methods. In Proceedings of the Advances in Neural Information Processing Systems 2019, Vancouver, BC, Canada, 8–14 December 2019; Volume 32.
18. Wu, J.; Chen, S.P.; Liu, X.Y. Efficient Hyperparameter Optimization through Model-Based Reinforcement Learning. *Neurocomputing* **2020**, *409*, 381–393. [CrossRef]
19. Holzinger, A.; Malle, B.; Saranti, A.; Pfeifer, B. Towards Multi-Modal Causability with Graph Neural Networks Enabling Information Fusion for Explainable AI. *Inf. Fusion* **2021**, *71*, 28–37. [CrossRef]
20. Barredo Arrieta, A.; Díaz-Rodríguez, N.; Del Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; Garcia, S.; Gil-Lopez, S.; Molina, D.; Benjamins, R.; et al. Explainable Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. *Inf. Fusion* **2020**, *58*, 82–115. [CrossRef]
21. Gunning, D.; Stefik, M.; Choi, J.; Miller, T.; Stumpf, S.; Yang, G.Z. XAI-Explainable Artificial Intelligence. *Sci. Robot.* **2019**, *4*. [CrossRef] [PubMed]
22. Gunning, D.; Aha, D. DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Mag.* **2019**, *40*, 44–58.
23. Castelvecchi, D. Can We Open the Black Box of AI? *Nature* **2016**, *538*, 20–23. [CrossRef]
24. Ribeiro, M.T.; Singh, S.; Guestrin, C. "Why Should i Trust You?" Explaining the Predictions of Any Classifier. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 1135–1144.
25. Zhang, Q.; Yang, Y.; Ma, H.; Wu, Y.N. Interpreting Cnns via Decision Trees. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 6254–6263.
26. Zhang, Y.; Tino, P.; Leonardis, A.; Tang, K. A Survey on Neural Network Interpretability. *IEEE Trans. Emerg. Top. Comput. Intell.* **2021**, *5*, 726–742. [CrossRef]
27. Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; Torralba, A. Learning Deep Features for Discriminative Localization. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2921–2929.
28. Selvaraju, R.R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *Int. J. Comput. Vis.* **2020**, *128*, 336–359. [CrossRef]
29. Sundararajan, M.; Taly, A.; Yan, Q. Axiomatic Attribution for Deep Networks. In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, Australia, 6–11 August 2017; Volume 7, pp. 5109–5118.
30. Shrikumar, A.; Greenside, P.; Kundaje, A. Learning Important Features through Propagating Activation Differences. In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, Australia, 6–11 August 2017; Volume 7, pp. 4844–4866.
31. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning Internal Representations by Error Propagation. In *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*; Morgan Kaufmann Publishers: Burlington, MA, USA, 1988; pp. 399–421. [CrossRef]
32. Gulcehre, C.; Moczulski, M.; Bengio, Y. ADASECANT: Robust Adaptive Secant Method for Stochastic Gradient. *arXiv* **2014**, arXiv:1412.7419.
33. Ruder, S. An Overview of Gradient Descent Optimization Algorithms. *arXiv* **2016**, arXiv:1609.04747.
34. Sutskever, I.; Martens, J.; Dahl, G.; Hinton, G. On the Importance of Initialization and Momentum in Deep Learning. In Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 17–19 June 2013; pp. 2176–2184.
35. Su, W.; Boyd, S.; Candès, E.J. A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights. *J. Mach. Learn. Res.* **2016**, *17*, 1–43.
36. Bubeck, S.; Lee, Y.T.; Singh, M. A Geometric Alternative to Nesterov's Accelerated Gradient Descent. *arXiv* **2015**, arXiv:1506.08187.
37. Duchi, J.; Hazan, E.; Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In Proceedings of the 23rd Annual Conference on Learning Theory (COLT 2010), Haifa, Israel, 27–29 June 2010; Volume 12, pp. 257–269.
38. Zeiler, M.D. ADADELTA: An Adaptive Learning Rate Method. *arXiv* **2012**, arXiv:1212.5701.
39. Dauphin, Y.N.; De Vries, H.; Bengio, Y. Equilibrated Adaptive Learning Rates for Non-Convex Optimization. In Proceedings of the Advances in Neural Information Processing Systems 2015, Montreal, QC, Canada, 7–12 December 2015; pp. 1504–1512.
40. Kingma, D.P.; Ba, J.L. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015.
41. Dozat, T. Incorporating Nesterov Momentum into Adam. In Proceedings of the ICLR Workshop, San Juan, Puerto Rico, 2–4 May 2016; pp. 2013–2016.
42. Hoseini, F.; Shahbahrami, A.; Bayat, P. AdaptAhead Optimization Algorithm for Learning Deep CNN Applied to MRI Segmentation. *J. Digit. Imaging* **2019**, *32*, 105–115. [CrossRef] [PubMed]
43. Yang, J.; Yang, G. Modified Convolutional Neural Network Based on Dropout and the Stochastic Gradient Descent Optimizer. *Algorithms* **2018**, *11*, 28. [CrossRef]
44. Engl, H.W.; Ramlau, R. *Regularization of Inverse Problems*; Springer: Amsterdam, The Netherlands, 2015; ISBN 978-0-7923-6140-4.
45. Ba, L.J.; Frey, B. Adaptive Dropout for Training Deep Neural Networks. *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 3084–3092.
46. Baldi P, S.P. BEATTY Understanding Dropouts. In Proceedings of the Advances in Neural Information Processing Systems 2013, Lake Tahoe, NV, USA, 5–10 December 2013; Volume 26.

47. Antonellis, G.; Gavras, A.G.; Panagiotou, M.; Kutter, B.L.; Guerrini, G.; Sander, A.C.; Fox, P.J. Shake Table Test of Large-Scale Bridge Columns Supported on Rocking Shallow Foundations. *J. Geotech. Geoenviron. Eng.* **2015**, *141*, 04015009. [CrossRef]
48. Jiongming, S.U.; Hongfu, L.I.U.; Fengtao, X.; Jianzhai, W.U.; Xingsheng, Y. Survey of Interpretation Methods for Deep Neural Networks. *Comput. Eng.* **2020**, *46*.
49. Dumoulin, V.; Visin, F. A Guide to Convolution Arithmetic for Deep Learning. *arXiv* **2016**, arXiv:1603.07285.
50. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *ICML Work. Deep Learn. Audio, Speech Lang. Process.* **2013**, *30*, 1.
51. Klambauer, G.; Unterthiner, T.; Mayr, A.; Hochreiter, S. Self-Normalizing Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 972–981.
52. Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv* **2017**, arXiv:1706.02677.
53. Keskar, N.S.; Nocedal, J.; Tang, P.T.P.; Mudigere, D.; Smelyanskiy, M. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, 24–26 April 2017.
54. Wang, H.; Bengio, S. *The MNIST Database of Handwritten Upper-Case Letters*; IDIAP: Martigny, Switzerland, 2002.
55. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification. In Proceedings of the IEEE International Conference on Computer Vision 2015, Las Condes, Chile, 11–18 December 2015; pp. 1026–1034.
56. Hoffer, E.; Hubara, I.; Soudry, D. Train Longer, Generalize Better: Closing the Generalization Gap in Large Batch Training of Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 1732–1742.
57. Singh, S.; Shrivastava, A. EvalNorm: Estimating Batch Normalization Statistics for Evaluation. In Proceedings of the IEEE International Conference on Computer Vision 2019, Seoul, Korea, 27 October–2 November 2019; pp. 3632–3640.