Journal of Electronics and Information Science (2023)
Clausius Scientific Press, Canada

DOI: 10.23977/jeis.2023.080306
ISSN 2371-9524 Vol. 8 Num. 3

Research on Residual Convolutional Neural Network for Handwritten Digit Recognition

Sizhe Zou*

Institute of Computer and Information Technology, Beijing Jiaotong University, Beijing, 100044, China
**Corresponding author: 21722080@bjtu.edu.cn*

Keywords: Residual Convolution; Handwritten Digit Recognition; Neural Networks

Abstract: The technology of handwritten digit recognition has been widely applied in various situations and has significant practical significance. However, the morphological features of handwritten numbers are very complex, and achieving accurate recognition of handwritten numbers relies on efficient and accurate recognition techniques. This article proposes a residual convolutional network model to address the issues of inaccurate feature extraction and weak model generalization ability in convolutional neural networks. By introducing residual blocks into the network, the problem of vanishing and exploding network gradients is effectively eliminated. At the same time, the Batch Normalization and Dropout layers are introduced to accelerate the network training process and reduce the risk of overfitting. Finally, the k-fold cross validation method was used to select the optimal parameter configuration of the model. The experimental results show that residual convolutional neural networks have the characteristics of high recognition accuracy and strong model generalization ability.

1. Introduction

Handwritten digit identification, categorization, and processing technology will replace manual extraction of digital information in a variety of industries, considerably enhancing productivity as it serves as the core and key of digital automation systems. Research on handwritten digit recognition, however, has the potential to evaluate and validate certain fresh hypotheses that have great theoretical relevance for challenges like Chinese and English character recognition. For the recognition of handwritten numerals, several academics have developed many recognition algorithms, including the BP neural network, the self-coding network, the convolutional neural network, etc. There are still some issues that affect recognition performance and result in low accuracy, such as the uncertainty of picture noise interference, despite the gradual increase in recognition rate and improvement in model performance; number recognition differs from text recognition in that it lacks context and can only be performed on the character itself, without the aid of any other recognition techniques; unable to strike a balance between speed and accuracy[1-3].

Zhou proposed to extract the curvature features of character contours. This method uses a "17/8/2" Back Propagation (BP) neural network for recognition. Due to the similar curvature of some numbers (such as "0" and "8", "2" and "4", etc.), the recognition rate is not too high (less than

95%) [4]. To solve the problems of long training time and local optimization in BP neural networks, scholars such as Wei Henghua proposed an improved genetic algorithm and used it to optimize the weights and thresholds of artificial neural networks. Based on this algorithm, a "256/16/10" BP neural network was constructed and effectively trained on the USPS handwritten digit sample set [5]. Liu Yang and other researchers used variable step method and Newton method to improve the BP algorithm, which improved the convergence speed of the network. The network convergence speed of this method is significantly faster than other improved algorithms. On this basis, the BP neural network recognition model will be applied to the digital recognition system. Although this method improves the algorithm speed, it requires more memory storage space, and the recognition rate of handwritten digits has not yet reached a very high level [6]. Yang et al proposed an online incremental learning algorithm based on support vector machines. This method calls the LIBSVM classifier training function and sample recognition function, and retrains the classifier with unrecognized samples as incremental data. The experimental results show that incremental training can improve sample training speed and improve the accuracy of handwritten digit recognition while taking into account the appearance of new input samples [7]. Convolutional neural networks-based handwritten digit recognition was proposed by Li Sifan and Gao Faqin. A better convolutional neural network model is created, and the MNIST character library is used to test the new model. The outcomes demonstrate the simplicity of the enhanced network topology, the reduced preprocessing workload, the great scalability, the quick recognition speed, and the high recognition rate. The recognition performance is noticeably better than the old methods, and it can successfully prevent the network from overfitting [8]. In summary, for the purpose of recognizing handwritten digits, it is frequently challenging for a single recognition model to accomplish both speed and high accuracy, whether utilizing BP neural networks or the support vector machine SVM algorithm. Traditional approaches still rely on hand-extracting sample features, and statistical features-based identification algorithms struggle to identify characters in an image when their shapes are similar. As a result, feature extraction is a primary area for improvement. Traditional artificial neural networks' design and implementation frequently rely too heavily on experience and generalization performance, making it difficult to guarantee the output would be optimal when used for actual handwritten digit recognition [9]. As a result, strengthening generalization capability and refining the network model are also important areas of advancement.

2. Residual neural network

2.1 Deep residual learning

Deep convolutional networks have made breakthrough progress in image classification tasks. In recent years, research has shown that the depth of neural networks has a crucial impact on classification performance. The network models that have achieved good results on the ImageNet dataset are all based on deep neural network models. As the depth of the network increases, a subsequent problem is gradient dispersion. The use of normalization initialization solves this problem to some extent and can make the number of stacked layers reach tens of layers [10].

As shown in Figure 1, set the input to x. Assume that the ideal map is f(x), which is the input to the activation function above Figure 1. The portion in the dashed box only needs to fit the residual mapping f(x) - x related to the identity mapping. Residual mapping is often easier to optimize in practice. Using identity mapping as the ideal mapping f(x). Simply set the weight and deviation parameters of the weighted operations (such as affine) above the dashed box in Figure 1 to 0, and then f(x) is an identity map. In practice, when the ideal mapping f(x) is very close to the identity mapping, the residual mapping is also easy to capture the subtle fluctuations of the identity mapping. Figure 1 is also the basic block of ResNet, which is the residual block. In residual blocks, input can propagate

faster forward through cross layer data lines. This short connection form neither introduces new parameters nor increases computational complexity [11].

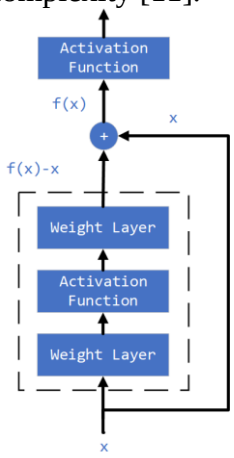

Figure 1: The structure of residual block

2.2 Batch normalization layer

Generally speaking, the batch normalization layer is effective enough for data standardization preprocessing in shallow models. As the model training progresses, it is difficult for the output near the output layer to undergo significant changes when the parameters in each layer are updated. However, for deep neural networks, even if the input data has been standardized, updating the model parameters during training is still more likely to cause drastic changes in output near the output layer. The instability of numerical calculations often makes it difficult for us to train effective depth models. The proposal of batch normalization layer is precisely to address the challenge of deep model training. During model training, the batch normalization layer utilizes the mean and standard deviation on small batches to continuously adjust the intermediate output of the neural network, thereby making the values of the entire neural network output in the middle of each layer more stable [12].

(1) Batch normalization of fully connected layers

Usually, we place the batch normalization layer between the affine transformation and the activation function in the full connection layer. Let the input of the full connection layer be $\boldsymbol{u}$, the weight parameter and deviation parameter be $\boldsymbol{W}$ and $\boldsymbol{b}$, respectively, and the activation function be ϕ. Set the operator for batch normalization to BN. Then, the output of the full connection layer using batch normalization is $\phi(\mathrm{BN}(\boldsymbol{x}))$, where the batch normalization input $\boldsymbol{x}$ is obtained by affine transformation:

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{u} + \boldsymbol{b}. \tag{1}$$

At the same time, consider a small batch consisting of m samples, and the output of affine transformation is a new small batch $\mathfrak{B} = \{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$. They are the inputs of the batch normalization layer. For any sample $\boldsymbol{x}^{(i)} \in \mathbb{R}^d, 1 \le i \le m$ in small batch $\mathfrak{B}$, the output of the batch normalization layer is also a d-dimensional vector:

$$\boldsymbol{y}^{(i)} = \mathrm{BN}(\boldsymbol{x}^{(i)}), \tag{2}$$

and obtain it from the following steps. Firstly, calculate the mean and variance for small batch $\mathfrak{B}$:

$$\boldsymbol{\mu}_{\mathfrak{B}} \leftarrow \frac{1}{\mathrm{m}} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}, \tag{3}$$

$$\boldsymbol{\sigma}_{\mathfrak{B}}^{2} \leftarrow \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\mathfrak{B}})^{2}. \tag{4}$$

The square calculation is based on the element. Next, standardize $\boldsymbol{x}^{(i)}$ using square root by element and division by element:

$$\hat{\boldsymbol{x}}^{(i)} \leftarrow \frac{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\mathfrak{B}}}{\sqrt{\boldsymbol{\sigma}_{\mathfrak{B}}^{2} + \varepsilon}}, \tag{5}$$

here is a very small constant $\varepsilon > 0$, ensuring that the denominator is not greater than 0. On the basis of the above standardization, the batch normalization layer introduces two learnable model parameters, the scale parameter $\boldsymbol{\gamma}$ and the shift parameter $\boldsymbol{\beta}$. These two parameters have the same shape as $\boldsymbol{x}^{(i)}$ and are both d -dimensional vectors. They are calculated by element multiplication (symbol $\odot$) and addition with $\hat{x}^{(i)}$, respectively:

$$\boldsymbol{y}^{(i)} \leftarrow \boldsymbol{\gamma} \odot \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}. \tag{6}$$

we have obtained the batch normalized output $\boldsymbol{y}^{(i)}$ of $\boldsymbol{x}^{(i)}$. Note that the learnable stretching and offset parameters reserve the possibility of not normalizing $\boldsymbol{x}^{(i)}$ in batches: at this point, only $\boldsymbol{\gamma} = \sqrt{\boldsymbol{\sigma}_{\mathfrak{B}}^{2} + \varepsilon}$ and $\boldsymbol{\beta} = \boldsymbol{\mu}_{\mathfrak{B}}$ need to be learned. If batch normalization is not beneficial, theoretically, the learned model can be avoided using batch normalization [13].

(2) Batch normalization of convolutional layers

For convolution layer, batch normalization occurs after convolution calculation and before activation function is applied. If the convolution calculation outputs multiple channels, we need to perform batch normalization on the outputs of these channels separately, and each channel has independent stretching and offset parameters, all of which are scalars. Set m samples in a small batch. On a single channel, assume that the height and width of the convolution calculation output are p and q, respectively. We need to perform batch normalization on m×p×q elements in this channel simultaneously. When performing standardized calculations on these elements, we use the same mean and variance, that is, the mean and variance of the m×p×q elements in the channel.

2.3 Residual neural network

ResNet follows the design of VGG's full 3×3 convolutional layers. As shown in Figure 2, there are first two 3×3 convolutional layers with the same number of output channels in the residual block. Each convolution layer is followed by a batch normalization layer and ReLU activation function. Then we will skip the two convolution operations and directly add the input to the last ReLU activation function. The implementation of residual blocks is as follows. It can set the number of output channels, whether to use an additional 1×1 convolutional layers to modify the number of channels, and the stride of the convolutional layers. ResNet follows the 7×7 convolutional layer with 64 output channels and a step of 2 with a maximum pooling layer of 33 with a step of 2. ResNet adds a batch normalization layer after each convolutional layer. ResNet uses four modules composed of residual blocks, each module using several residual blocks with the same number of output channels. Each subsequent module doubles the number of channels from the previous module in the first residual block and halves the height and width [14].

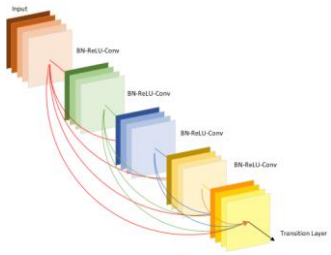

Figure 2: ResNet

3. Experimental Results and Discussion

3.1 Experimental Data and Environment

The MNIST handwritten digit dataset is from the National Institute of Standards and Technology in the United States. It consists of two types: a training set and a testing set, written and produced by a total of 250 high school students and Census Bureau staff, with students and staff accounting for 50% each. The training set contains 60000 training samples, and the test set contains 10000 test samples. The dataset is divided into four parts: training image set, training label set, testing image set, and testing label set. Each MNIST image is a digitized image of a single handwritten 0-9 numeric character. Each image is 28×28 pixel size. A pixel value of 0 represents white, a pixel value of 255 represents black, and the middle pixel value represents grayscale. Part of the training set samples are shown in Figure 3.

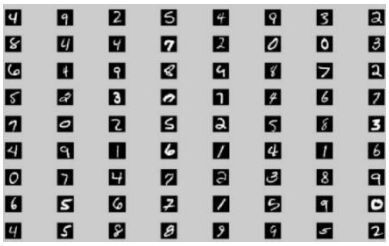

Figure 3: Partial samples from the MNIST dataset

Due to the fact that the validation dataset does not participate in model training, it is too luxurious to reserve a large amount of validation data when the training data is insufficient. One improvement method is k-fold cross validation. In k-fold cross validation, we divide the original training dataset into k non-overlapping sub-datasets, and then we perform k-fold model training and validation. Each time, a sub dataset is used to validate the model and train the model using other k-1 sub-datasets. In this k training and validation sessions, the sub-datasets used to validate the model were different each time. Finally, we averaged training error and validation error for these k times. We used the deep residual convolutional neural network shown in Figure 4 for this experiment.

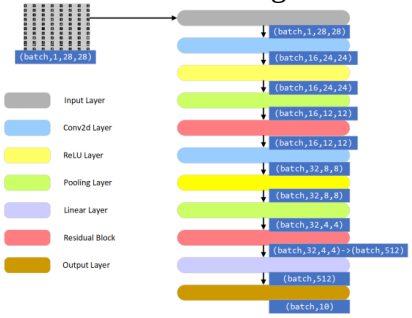

Figure 4: The network model used in the experiment

3.2 Handwritten Digit Prediction Experiment

First, initialize all weight parameters using the Xavier method, set the initial values of all deviation parameters to 0, train 60000 training samples in the MNIST dataset, and adjust model parameters through back-propagation. Afterwards, transfer the training set to the trained model and observe the classification results. In the RCN model used in this article, when epoch=10, the gradient descent small batch size is 64, the learning rate is 0.01, and the momentum method parameter is chosen as 0.5. As follow, the results of the prediction experiment are shown Table 1 and Figure 5.

Table 1: Predicting experimental results data

Epoch	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
1	4.691547	91	14.577732	97
2	3.490832	97	9.012899	98
3	2.229128	98	8.367525	98
4	1.599396	98	6.592390	98
5	1.061613	98	6.336032	98
6	1.355022	98	5.778624	98
7	0.995296	98	5.065052	98
8	0.968919	99	4.839839	98
9	0.640158	99	5.119769	99
10	1.054762	99	5.028024	98

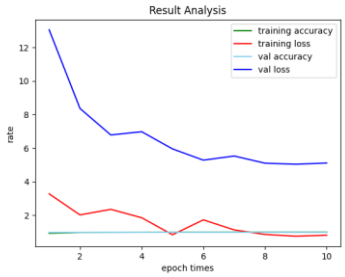

Figure 5: Predicting experimental results

According to the chart data, the training error gradually decreases with the iteration cycle, with a slight rebound in the sixth cycle and then continuing to decrease. The generalization error decreased significantly in the first three iteration cycles, and then continued to decline gently to about 5. The training recognition rate rapidly increased from 91% to over 97% in the first three iteration cycles, while the validation recognition rate slowly increased and remained above 97% from the beginning, indicating that the model has good generalization ability.

The training error and generalization error of the model used in this paper steadily decline when the iteration cycle gradually increases. After three iteration cycles, the training and testing correct rate are both stable at more than 98%, with high accuracy. The training process is relatively stable, with no significant error jumps. This experiment verifies the efficiency of the RCN model, which can effectively complete classification and prediction tasks.

3.3 Sensitivity analysis of hyperparameter

First, initialize all weight parameters using the Xavier method, set the initial values of all deviation parameters to 0, train 60000 training samples in the MNIST dataset, and adjust model parameters through back-propagation. Afterwards, transfer the training set to the trained model and observe the classification results.

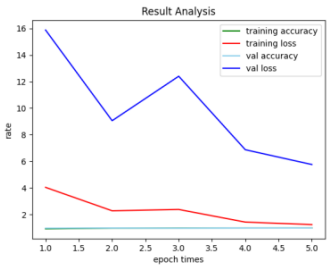

Figure 6: Epoch=5

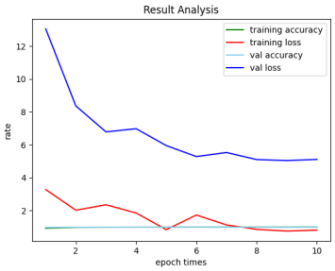

Figure 7: Epoch=10

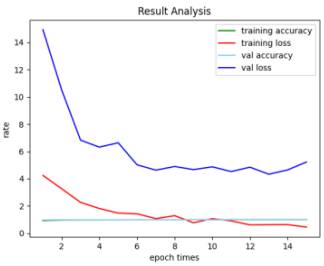

Figure 8: Epoch=15

As shown in Figure 6, when epoch=5, the validation error in the first two cycles rapidly decreases, and the validation error in the third iteration cycle rebounds, but continues to steadily decrease thereafter. When epoch=10, as shown in Figure 7, the validation error in the first three cycles rapidly decreases, and the third cycle also reaches the platform stage, before stabilizing again. When epoch=15, as shown in Figure 8, the situation is similar to Figure 7. The experimental results show that, with the constant change of epoch setting, the training correct recognition rate finally reaches 99%, and the verification correct recognition rate finally reaches 98%. Therefore, the sensitivity of the hyperparameter iteration cycle is low. When it increases or decreases, the model performance does not change significantly.

Figure 9: Batch_Size=16

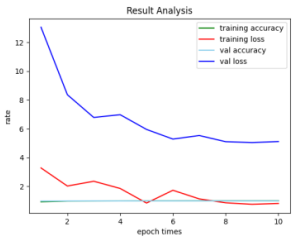

Figure 10: Batch_Size=6

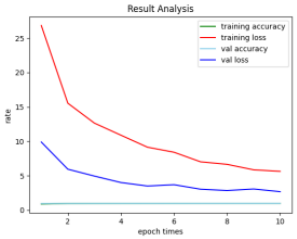

Figure 11: Batch_Size=128

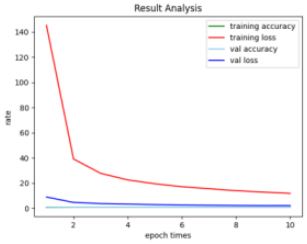

Figure 12: Batch_Size=256

Secondly, control the iteration period, keep the learning rate and momentum method parameters constant, and take small batches as 16, 64, 128, and 256, respectively. When batch_size=16, as shown in Figure 9, the validation error has slightly increased in the third, fifth, seventh, and ninth cycles, but the overall trend is decreasing with the iteration cycle, which is not stable enough. When batch_size=64, as shown in Figure 10, the validation error rapidly decreased in the first three cycles, and also reached the platform stage in the third cycle, before stabilizing again. When batch_size=128, as shown in Figure 11, the training error is greater than the verification error. At the beginning, the training error is very large, which drops rapidly in the first two cycles, and then the two errors continue to decline gently. The model shows excellent generalization performance. Generalization error eventually converges to about 5, but the training error is higher than the previous super parameter selection. When batch_size=256, as shown in Figure 12, the situation is similar to Figure 11, but the training error decreases more rapidly in the first two cycles and gradually decreases

thereafter. The experimental results show that when the size of hyperparameter small batch changes, the training recognition rate and test recognition rate of the model on the MNIST dataset show almost no change.

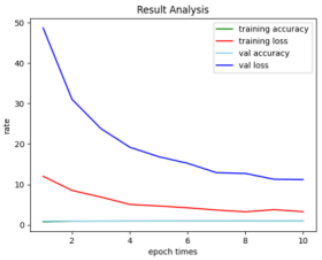

Figure 13: Learning_Rate=0.005

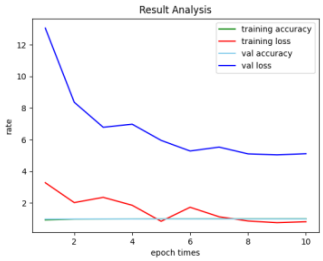

Figure 14: Learning_Rate=0.01

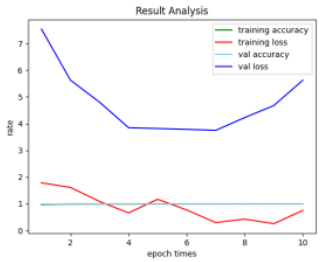

Figure 15: Learning_Rate=0.1

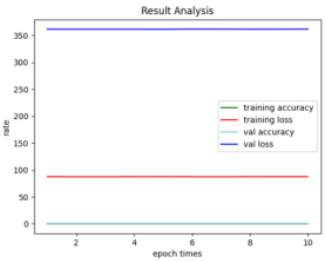

Figure 16: Learning_Rate=0.3

Then, control the iteration period, keep the parameters of the small batch and momentum methods unchanged, and take the learning rates as 0.005, 0.01, 0.1, and 0.3, respectively. When learning_rate=0.005, as shown in Figure 13, the training error and verification error both decrease with the iteration cycle, but the training error is always high, the generalization error is far greater than the training error, and the model has a certain degree of overfitting. When learning_rate=0.01, as shown in Figure 14, the validation error rapidly decreased in the first three cycles, and also reached the platform stage in the third cycle before stabilizing again. When learning_rate=0.1, as shown in Figure 15, the training error rapidly decreases in the first four cycles, followed by a brief plateau period, and then slightly increases. When learning_rate=0.3, as shown in Figure 16, the training error is very large, the generalization error is very large, the network model is divergent, and the classification task cannot be completed. The experimental results show that the performance of the model changes greatly when the super parameter learning rate changes. When the learning rate decreases, the model appears over fitting phenomenon, the generalization error is large, and the training and test correct recognition rates are low; When the learning rate increases, the model generalization error begins to increase until the model diverges and the classification task cannot be carried out.

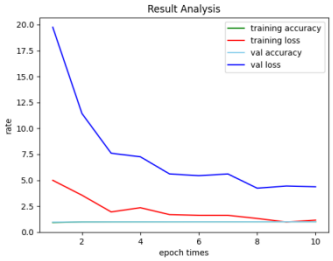

Figure 17: M=0.3

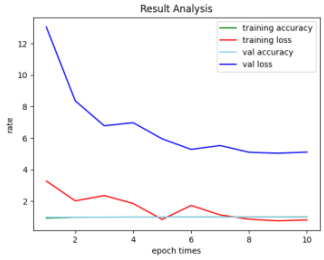

Figure 18: M=0.5

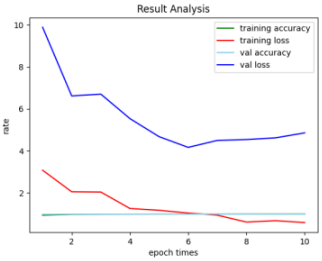

Figure 19: M=0.7

Finally, control the iteration period, with the small batch and learning rate parameters unchanged,

and take the momentum method parameters as 0.3, 0.5, and 0.7, respectively. When m=0.3, as shown in Figure 17, the validation error in the first three cycles rapidly decreases, and then continues to steadily decrease, resulting in a stable training process. When m=0.5, as shown in Figure 18, the validation error in the first three cycles decreased rapidly, and the third cycle also reached the plateau period, before stabilizing again. When m=0.7, as shown in Figure 19, the validation error rapidly decreased in the first two cycles, reached the plateau period in the third cycle, and then steadily decreased, with a slight rebound starting from the seventh cycle. Therefore, the sensitivity of the hyperparameter momentum method parameter is low, and the model performance does not change significantly when increasing or decreasing. To sum up, it can be found that hyperparameter such as small batch size and learning rate are more sensitive, while iteration period and momentum method hyperparameter are not sensitive to numerical changes.

3.4 Hyperparameter cross validation experiment

Through the above experiments, we have explored the sensitivity of model hyperparameters, and we would like to further explore the optimal combination of hyperparameters. By applying k-fold cross validation, the training set is evenly divided into 10 sets that do not intersect with each other, and different hyperparameters are selected to transform different sensitive hyperparameters. This experiment verifies different combinations of hyperparameters by selecting small batches and learning rates. Based on the results of cross validation, we obtained the average training error and average generalization error (excluding singular values) for different small batches with learning rates ranging from 0.005 to 0.195 and step sizes of 0.005, as shown in Table 2:

Table 2: Cross validation experimental results data

Batch_Size	Mean Train_Loss	Mean Val_Loss
32	Divergence	Divergence
64	1.999409	4.658475
96	0.283355	0.956955
128	0.581042	0.583492
160	5.957884	2.987078
192	0.107769	0.402529
224	1.013017	0.314624
256	1.321170	0.481471

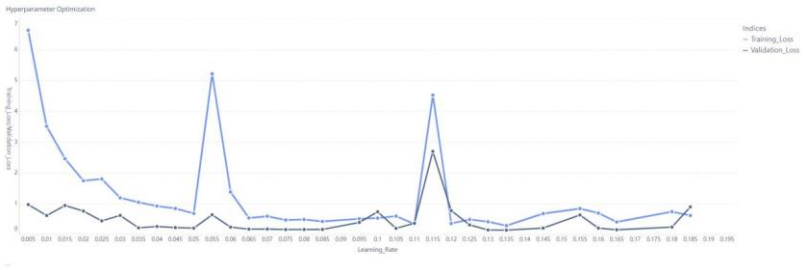

Figure 20: Cross validation results at Batch_Size=256

Figure 20 shows the experimental results of learning rate, training error, and generalization error when the small batch size is 256. From the chart analysis, selecting too small a batch (such as 32) can easily cause overfitting, and both types of errors are relatively high. When the small batch size is 160 and 256, the average generalization error has slightly increased, but overall it shows a downward trend. When the small batch size is 256, the generalization error is smaller than the training error, and the model exhibits extremely strong generalization performance. The learning rate ranges from 0.005 to 0.05, and the training error rapidly decreases. There is a very small rebound at a learning

rate of 0.025, and then continues to decline; Singular values appear at learning rates of 0.055 and 0.115. Based on the above results, while observing a reasonable decrease in training error, the hyperparameter combination with the lowest generalization error was selected. Finally, an iteration period of 10, a small batch size of 256 for gradient descent, a learning rate of 0.135, and a momentum method parameter of 0.5 were selected.

4. Conclusions

This article proposes a deep convolutional neural network model based on residual module to address the risk of overfitting caused by insufficient sample size when applying deep network models to handwritten digit recognition, as well as the problem of gradient vanishing and explosion caused by excessively deep network layers. The experimental results show that the training recognition rate rapidly increased from 91% to over 97% in the first three iteration cycles, while the validation recognition rate slowly increased and remained above 97% from the beginning, indicating that the model has good generalization ability. This experiment proves that hyperparameters such as small batches and learning rates are more sensitive, while iteration period and momentum method hyperparameters are not sensitive to numerical changes. The hyperparameter cross validation experiment has proven that selecting too small a batch can easily cause overfitting, and if the learning rate increases, the network is prone to divergence. By testing the sensitivity of each hyperparameter in the network model and selecting the optimal combination of hyperparameters, a deep convolutional residual network model with strong generalization ability and high positive recognition rate can be trained.

References

[1] Huang Rui, Lu Xuming, Wu Yilin. Cursive number recognition and application based on Tensor Flow deep learning [J]. Electron Technology Applications, 2018, 44(10):6-10. DOI:10.16157/j.issn.0258-7998.182249.

[2] He Ping, Liu Ziyan. Handwritten digit recognition based on improved multi-layer perceptron [J]. Communication Technology, 2018, 51(09):2075-2080.

[3] Tan Shuai. Research on handwritten digit recognition based on deep residual networks [D]. Xi'an University of Electronic Science and Technology, 2019.

[4] Chen Yan, Li Yangyang, Yu Le, Wang Yao, Wu Chao, Li Yangguang. Cursive number recognition system based on convolutional neural network [J]. Microelectronics and computer, 2018, 35(02):71-74. DOI:10.19304/j. cnki. issn1000-7180. 2018.02.015.

[5] Song Xiaoru, Wu Xue, Gao Song, Chen Chaobo. Simulation research on handwritten digit recognition based on deep neural networks [J]. Science Technology and Engineering, 2019, 19(05):193-196.

[6] Bu Lingzheng, Wang Hongdong, Zhu Meiqiang, Dai Wei. Multi source number recognition algorithm based on improved convolutional neural network [J]. Computer Application, 2018, 38(12):3403-3408.

[7] Huang Yitian, Chen Zhitong. Recognition of handwritten digits based on convolutional neural network in the Pytorch framework [J]. Electronic Technology and Software Engineering, 2018(19):147.

[8] Hu Junping, Fu Kexue. Research on handwritten digit recognition based on improved KNN algorithm [J]. Journal of Wuhan University of Technology (Information and Management Engineering Edition), 2019, 41(01):22-26.

[9] Yang Gang, He Dongge, Dai Lizhen. Research on handwritten digit recognition based on CNN and particle swarm optimization SVM [J]. Journal of East China Jiaotong University, 2020, 37(04):41-47. DOI:10. 16749/j. cnki. Jecjtu .2020.04.007.

[10] Lv Hong. Design of handwritten digit recognition system based on convolutional neural network [J]. Intelligent Computers and Applications, 2019, 9(02):54-56+62.

[11] He K., Zhang X., Ren S., & Sun J. Deep residual learning for image recognition [J]. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2019:770-778.

[12] Kulkarni S.R. and Rajendran B. Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization [J]. Neural networks, 2018, 103:118-127. DOI:10.1016/j.neunet. 2018.03.019.

[13] Zhang A. et al. Dive into Deep Learning [M]. 2021. doi:10.48550/arxiv.2106.11342.

[14] Kusetogullari H. et al. Digitnet: A deep handwritten digit detection and recognition method using a new historical handwritten digit dataset [J]. Big data research, 2021, 23:100-182. DOI:10.1016/j.bdr. 2020.100182.