Journal of Computers Vol. 31 No. 1, 2020, pp. 46-56
doi:10.3966/199115992020023101004

Deeply Integrated Convolutional Neural Networks

Daojun Liang[1], Xiuping Wang[1], Xiaohui Ju[1], Feng Yang[1,2*]

[1] School of Information Science and Engineering, Shandong Normal University, Jinan 250014, Shandong Province, China
{liangdaojun, wangxiuping, juxiaohui}@stu.sdnu.edu.cn

[2] Institute of Data Science and Technology, Shandong Normal University, Jinan 250014, Shandong Province, China
yangfeng@sdnu.edu.cn

Received 20 April 2018; Revised 29 August 2018; Accepted 5 October 2018

Abstract. The ensemble learning system based on neural network requires a large number of networks as the basic classifier, which makes the parameters and calculations of the system increase sharply. Integrating the neural network in depth can not only reduce the parameters and calculations of the network, but also improve the overall network performance. In this paper, a deeply integrated convolutional neural network (DICNN) was proposed, and several different integration methods were proposed for integrated learning of DICNN. The Mixup is used to train the DICNN because it uses multiple samples for training and it can be better combined with DICNN. A series of ablation experiments were done to prove that the training method of Mixup is equivalent to a regularization and data augmentation. Therefore, a different multi-sample training method as variations of the Mixup (Mixup-XL) can be used to train the DICNN.

Keywords: computer vision, deep learning

1 Introduction

Neural network ensemble is regarded as an engineering neural computing technology with broad application prospects, and has become a research hotspot in the field of machine learning and neural computing [1-2]. It can significantly improve the generalization ability of the learning system by training multiple neural networks and combining their conclusions [3-4]. The use of deep neural networks for ensemble learning will make the learning system have a huge amount of parameters and computation, which is also a challenge for existing ensemble algorithms. In this paper, a depth-based integrated convolutional neural network framework is proposed (DICNN). The architecture uses different depths of network blocks for ensemble learning, and each block structure is considered a basic classifier. The basic block structure can be integrated using different algorithms. For example, these base classifiers can be concatenated, added, maximized, and voted. DICNN is a method of neural network pseudo-ensemble learning, because its basic classifiers are not completely independent, but its various classifiers are not inseparable. DICNN can make multiple basic classifiers have the opportunity to participate in the final classification decision, which is beneficial to improve the generalization performance of the entire neural network. Fig. 1 shows the architecture of DICNN schematically.

* Corresponding Author

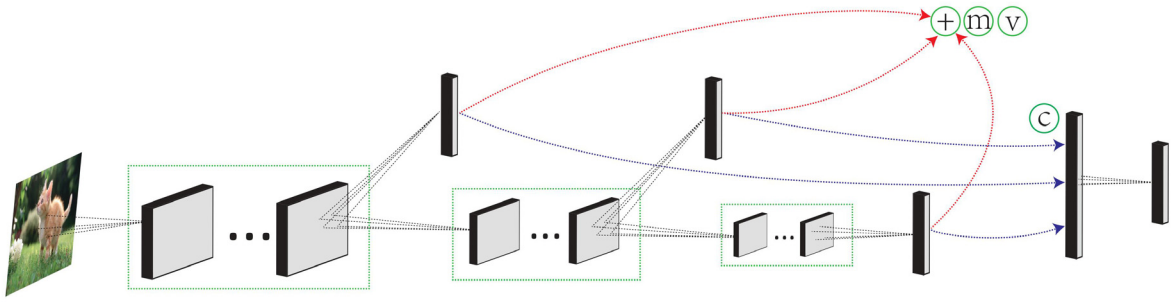


Fig. 1. The architecture of DICNN. The cube represents the convolutional layer, and the cuboid represents the fully connected layer. The symbol "c" represents the concatenation operation, and the symbols "+", "m", and "v" represent the addition operation, the maximize operation, and the voting operation, respectively.

Although the existing channel-fused neural network achieves implicit supervision, such as ResNets [5] and DenseNets [6], the network performance is lower than explicit supervising or integrating layers of different depths. Some experiments in this paper prove that DICNN can effectively improve the network with residual structure. The Mixup [7] method is used to train DICNN because it can take multiple samples as input at the same time, so that different loss functions can be used on the basic classifier at different depths. We also did some experiments to verify and explain why the Mixup is better than the Empirical Risk Minimization (ERM) [8] training method. These experiments prove that Mixup not only plays a role in data augmentation, but also plays a role in regularizing network parameters. A variant of the Mixup method (Mixup-XL) can be obtained by mixing the sample and the label separately using random values of different distributions. DICNN trained using the Mixup-XL method will have better generalization capabilities.

2 Related Work

Convolutional neural networks have made great progress in many fields, and the research of the network architecture has never stopped. There are a lot of networks that have achieved very good performance by applying new architectures. AlexNet [9] is the first to demonstrate the generalization ability of convolutional neural networks on large data. VGGNets [10] show that better performance can be achieved with smaller convolutional kernels and deeper layers. GoogLeNets [11] use different convolution kernels to establish more connections and more diverse representations between adjacent layers. ResNets [5] and Highway Networks [12] add the front layer information to the back layer through the bypass structure, which is more conducive to the backpropagation of the gradient, thus further deepening the depth of the network. ResNeXts [13] combine group convolution into ResNets [5], which perform split-transform-merge operations on features to improve network performance while reducing parameters. DenseNets [6] pass the features of each preceding layer to all of its subsequent layers to alleviate the vanishing/exploding gradient problem and to facilitate information fusion between layers.

Hansen and Salamon [1] pioneered the neural network ensemble method. They proved that the generalization ability of the neural network system can be significantly improved by simply training multiple neural networks and synthesizing the results. Maclin and Opitz [3] believe that neural network ensemble refers to multiple independently trained neural networks to learn and jointly determine the final output, and does not require the integrated network to learn the same subproblem. When neural network ensemble is used for regression estimation, the integrated output is typically generated by the output of each network by simple averaging or weighted averaging. Perrone and Cooper [14] believe that using a weighted average gives better generalization than a simple average. However, Opitz and Shavlik [15] believe that optimizing the weights will lead to overfitting, which will reduce the generalization ability of the ensemble. Therefore, they recommend using a simple average. Zhang et al. [16] use multiple neural networks to combine multiple predictions. Jabcobs [17] optimize each group of subnets to better handle an input subspace. However, Jimenez [18] does not use the linear combination method, but use some dynamic weights that vary with the degree of certainty of the individual network output to produce the final classification. Ueda [19] has chosen the best network for each output classification based on the

minimum classification error and then estimated the optimal linear weight to integrate the individual networks.

There are many ways to make the deep neural network achieve good generalization ability. Data augmentation is one of the most effective methods. Because it does not increase the parameters of the model and is easy to implement, it is widely used in deep neural network to control the complexity of the model and improve its generalization performance. For example, commonly used random clipping, horizontal flipping, adding a small amount of noise, and adjust the brightness, saturation, contrast of images. SamplePairing [20] and Mixup [7] have proven to be more generalizable than traditional training methods on many different datasets. In Mixup [7], a data augmentation method using a combination of two pictures in a training set is proposed. SamplePairing [20] randomly selects a sample in the training set to perturb the original sample (adding the two samples and then averaging them) and using the label of the original sample as the label of the new sample. Because different types of training samples are introduced during training, the neural network trained by the method has high training errors and losses, and the original samples need to be used to finetune the neural network or use a certain proportion of original samples in each batch. Mixup [7] uses a random value from the beta distribution as a weight to interpolate the two samples and their corresponding labels, respectively. The neural network trained in this way can achieve relatively low training errors and losses without the need for finetuning. This method trains a neural network on convex combinations of pairs of examples and their labels, which regularizes the neural network to favor simple linear behavior in-between training examples.

3 DICNN Architecture

In this section, how to divide the neural network into basic classifiers will first be introduced in Section 3.1. Then, the integrated methods for each basic classifier is proposed in Section 3.2. Finally, the loss function used to train DICNN is introduced in Section 3.3.

3.1 Basic Classifier

In order to integrate the neural network in depth, it is first necessary to divide the network into multiple basic classifiers. Generally speaking, a basic classifier can be divided every few fixed layers, and then the basic classifiers can be integrated into different methods to obtain an integrated learning system. For a neural network with a block structure, it is highly desirable to treat each block structure as a base classifier because these blocks have a complete feature structure as classification information.

When the block structure of a network is used as the basic classifier, it is first necessary to perform batch normalization (BN) [21] operation and rectified linear unit (Relu) [22] operation on its features, and then perform downsampling pooling on the feature map to obtain a one-dimensional fully connected layer. We record the basic classifier of DICNN as C_i, where i is the number of the classifier, its value increases with depth.

3.2 Ensemble method

As shown in Fig. 1, there are many integration methods available: the basic classifiers can be concatenated, added, maximized, voted, etc.

Concatenation. Combine all the basic classifiers to get a one-dimensional vector, then perform BN and Relu operations on the vector, and finally use a fully connected layer to match the number of sample categories. This method is called DICNN-C:

$$C_{final} = \begin{cases} f([C_1, C_2, \cdots, C_N]) \\ f(\sum_i^N C_i) \\ f(\prod_i^N C_i) \end{cases} \quad \textbf{(1)}$$

where the brackets "[ ]" represent the concatenation operation, and f represents a composite function containing BN, Relu, and fully connected layers. Eq. (1) shows that the final vector of the input

composite function f can be obtained by the concatenation operation, the addition operation, or the multiplication operation. We identify the various methods by adding suffixes. For example, C_{final} obtained by the concatenation operation is recorded as DICNN-C. The method of obtaining C_{final} by the addition operation is recorded as DICNN-CA. Similarly, the C_{final} obtained by the multiplication operation is recorded as DICNN-CM.

Addition. Add all the base classifiers directly as the final classifier. It should be noted that the number of each base classifier should be equal to the number of sample categories. This method is called DICNN-A:

$$\begin{aligned} C_{final} &= \sum_{i=1}^{N} \alpha_i C_i \\ s.t.\ &\sum_{i=1}^{N} \alpha_i = 1, \quad \alpha_i > 0 \end{aligned} \tag{2}$$

where α_i in represent the weight value of each base classifier.

Maximize. The classifier with the highest prediction probability among all the base classifiers is used as the final classifier. This method is called DICNN-M:

$$C_{final} = \max_i \{C_i\} \quad 1 \le i \le N \tag{3}$$

Voting. Relative majority voting rules are used to vote on the basic classifiers: a classification becomes the final result if and only if the classification has the largest number of base classifiers. This method is called DICNN-V:

$$C_{final} = \begin{cases} \sum_{i}^{N} I(C_i) & \text{iff. only one largest element exists} \\ \begin{cases} \sum_{i}^{N} C_i \\ \max_i \{C_i\} \end{cases} & otherwise. \end{cases} \tag{4}$$

where $I(C_i)$ represents the maximum probability category index in C_i. In Eq. (4), if there are two largest elements in the final vector, the final classifier will be set to a classifier that adds all the base classifiers or takes the highest probability among all the base classifiers. The method using only the voting operation is recorded as DICNN-V, and the method of combining the voting operation and the maximizing operation is recorded as DICNN-VM.

3.3 Training Method

Mixup. SamplePairing and Mixup are very similar to traditional data augmentation methods, but they are also different. SamplePairing adds other samples from the training set as noise to regularize the neural network during training. Mixup does not explicitly use the original sample training network, but the combination of the original sample. In some sense, Mixup is a smooth way of SamplePairing. Mixup is also a good regularization method. Our experiments show that Mixups actually train and predict two or more samples simultaneously. When training multiple samples at the same time, there are cases where multiple samples are confused with each other. The Mixup approach avoids the confusion between multiple samples, easily separating two different categories.

Mixup uses the convex combination of the original samples to train the neural network. Does the neural network make convex interpolation to the sample? Our experiments show that there is no connection between the way of samples combination and the way of labels combination. These combinations can come from different distributions, and neural networks can accurately capture these combinations. In other words, the neural network has the ability to learn and predict multiple sample categories at the same time.

The Mixup training method in the original paper is implemented by using a random value to weight the samples and their labels. Mixup can be formalized as:

$$\begin{aligned} &\tilde{x} = \lambda x_i + (1-\lambda)x_j \\ &\tilde{y} = \lambda y_i + (1-\lambda)y_j \\ &s.t.\quad 0 \le \lambda \le 1 \end{aligned} \tag{5}$$

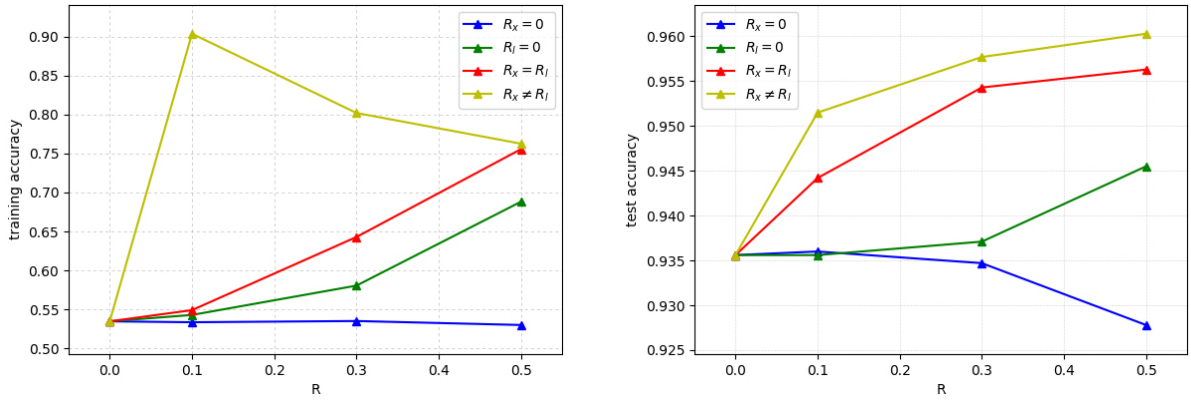
where (x_i, y_i) and (x_j, y_j) are two examples drawn at random from our training data, $\lambda \sim Beta(\alpha, \alpha)$ for λ in $[0,1]$. In the Mixup, training samples and their labels simultaneously reduce the λ or $1-\lambda$ times, implying this linear relationship. The signal strength scaled by λ or $1-\lambda$ times. Our experiments show that training samples and their labels do not simply have a linear relationship.

Mixup-XL. Let's take the sample's mixing ratio as λ_x and the sample label's mixing ratio as λ_l. Our empirical experiments show that the sample mixing ratio λ_x and the label mixing ratio λ_l do not have to be the same and do not have to come from the same distribution. We define the two ratios of λ_x and λ_l to be $[0.5-R_x, 0.5+R_x]$ and $[0.5-R_l, 0.5+R_l]$ respectively, where R_x, R_l in $[0, 0.5]$. When R_x is equal to R_l, we write them uniformly as R. Conversely, when we use R, it means R_x is equal to R_l. Fig. 2 analyzes the training accuracy of Mixup training methods for different values of R_x and R_l. In Fig. 2, $R_x = 0$, only R_l is changing. We call this training method Mixup-L. If $R_l = 0$, only R_x is changing, this training method is called Mixup-X. When $R_x = R_l$, this training method is the original Mixup. If R_x and R_l are all changing ($R_x \neq R_l$), and they obey different distributions. This trianing method is called Mixup-XL:

$$\begin{aligned} &\tilde{x} = \lambda_x x_i + (1-\lambda_x)x_j \\ &\tilde{y} = \lambda_l y_i + (1-\lambda_l)y_j \\ &s.t.\quad 0 \le \lambda_x, \lambda_l \le 1 \end{aligned} \tag{6}$$

where $\lambda_x \sim Beta(\alpha, \alpha)$ and $\lambda_l \sim Normal(m, v)$. Using different distributions of random values to separately weighted the training samples and their labels, and this method can also achieve the same effect as the Mixup. This shows that neural networks not only learn simple linear interpolation of input and output, but also learn interpolation between different distributions. However, the neural network will have higher training error and loss if either the input or the output of the fixed neural network is used (only the samples are weighted by a random value or the samples are weighted by random values). The generalization error of the Mixup is higher than that of the traditional training method.

Fig. 2 shows that the Mixup training method is quite different from the traditional training methods. In the traditional training method, the input and output are fixed, the training error can be reduced to 0. In Mixup, however, the training error is not reduced to 0 if the inputs and outputs are fixed, but this does not guarantee that the generalization error is higher than the generalization error of the network using the traditional method. The use of mixup has a higher loss than the traditional method, but it can achieve a lower generalization error, which shows that the two training methods are different in essence. This also shows that the neural network will become more clear the critical surface, sample confusion will be further eliminated. When $R_l = 0$, the training accuracy gradually increases with the increase of R_x, which is already higher than that of ERM training when $R_x = 0.5$. When $R_x = 0$, the performance of the network gradually declines, indicating that the simple regularization method does not work well after the data augmentation of the Mixup method is removed. It can be found that the performance of the Mixup-XL method is slightly higher than that of the Mixup method, which means that differently distributed samples and label combinations can achieve better results.



(a) (b)

Fig. 2. Training and testing accuracy of Mixup and its variants using the PreAct-ResNet-18 [23] network on CIFAR-10 dataset [24]. The x-axis represents the range of variation of the mixing ratio.

3.4 Loss Function

The difference with the traditional network is that DICNN has multiple base classifiers. We directly add the cross-entropy of the output of multiple base classifiers. The loss of the whole network is:

$$\begin{aligned} Loss &= \sum_{k=1}^{N} \beta_k \, E_{y\sim p(y)} [\log \; C_k \, (y)] \\ s.t. \quad & \sum_{k=1}^{N} \beta_k = 1 \end{aligned} \tag{7}$$

where β_k represent the weight value of loss function of the base classifier. If the Mixup or Mixup-XL method is used to train DICNN, the corresponding loss function can be formalized as:

$$\begin{aligned} Loss &= \sum_{k=1}^{N} \beta_k \, (E_{y_i\sim p(y_i)} [\log \; C_k \, (\lambda y_i)] \\ &\quad + E_{y_j\sim p(y_j)} [\log \; C_k \, ((1-\lambda) y_j)]) \\ s.t. \quad & \sum_{k=1}^{N} \beta_k = 1 \end{aligned} \tag{8}$$

where y_i and y_j represent the labels corresponding to the random sample x_i and x_j in the training set, and β_k represent the weight value of loss function of the base classifier.

4 Experiments

4.1 Dataset

The CIFAR-10 [24] dataset consists of 60000 32×32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

The CIFAR-100 [24] dataset has 100 classes, 500 training images and 100 testing images for each classes. Channel means are computed and subtracted in preprocessing. We also apply standard augmentation [5-6]: horizontal flipping and translation by 4 pixels are adopted in our experiments.

4.2 Training

For comparison purposes, we use the ResNet-56 [5] and DenseNet-BC-70 [6] network architecture for all datasets and set the same hyperparameters and training procedures for neural networks trained on CIFAR-10 [24] dataset. ResNet-56 [5] and DenseNet-BC-70 [6] are used as the basic network architecture.

All the networks are trained on two Tesla k80 GPUs using stochastic gradient descent (SGD). We use a weight decay of $1x10^{-4}$ and a Nesterov momentum [25] of 0.9 without dampening. The batch size on each GPU is set to 128 for 100 epochs. The initial learning rate is set to 0.1 and is divided by 10 at 40%, 60% and 80% of the total number of training epochs. The balance factor α in Eq. (2) is set to $\frac{1}{N}$, and the balance factor in Eq. (7) and Eq. (8) is set to $\frac{1}{N}$. The mean m and the variance v in Eq. (6) are set to 0.6 and 0.3 for λ_l, and the value of α is set to 1 for λ_x.

4.3 Fineturning Mixup Training Method

For networks using different Mixup training, further finetuning can improve the generalization performance of the single-category network. Note that in this finetuning process, multiple categories of input and output layers will be replaced by a single category.

We carried out experiments use ResNet-56 [5] on CIFAR-10 [24] and CIFAR-100 [24]. The ResNet-56 [5] is divided into 3 blocks, each of which is a residual block containing two convolutional layers. After each block, the feature size is halved and the number of channels is doubled. ResNet-56 [5] is used for all datasets and set the same hyperparameters and training procedures for neural networks trained on different datasets.

The results are shown in Table 1 and Table 2. From the Table 1, we can see that the single class precision of the network after the finetuning has a greater increase, which is because the confusion between multiple categories is eliminated. This makes the network better fit for a single class of samples. From Table 2, it can also be seen that the network performance after the finetuning has improved significantly, with an average of 3% improvement in accuracy. This shows that in the finetuning process, the Mixup method eliminates the confusion of other classification categories, so it increases the generalization ability of the network.

Table 1. Finetuning the ResNet-56 [5] using different training methods on CIFAR-10 [24]

Network	Mixup-X	Mixup-L	Mixup-C	Mixup-XL
Original	90.2%	90.8%	92.5%	92.8%
Finetuning	93.1%	93.5%	93.7%	94.1%

Table 2. Finetuning the ResNet-56 [5] using different training methods on CIFAR-100 [24]

Network	Mixup-X	Mixup-L	Mixup-C	Mixup-XL
Original	74.9%	75.2%	75.5%	76.1%
Finetuning	77.8%	78.9%	79.2%	79.8%

4.4 DICNN Performance

In this section, we use DenseNet-BC-70 [6] as the basic network architecture. The network has three block structures, each block structure consisting of 11 densely concatenated 1×1 and 3×3 convolutional layers. The transfer structure consists of a 1×1 convolutional layer and a pooled layer with a step size of 2, which is followed by all block structures but does not include the last block structure. DICNN added a basic classifier to each transfer structure of DenseNet-BC-70 [6], so DICNN has three classifiers. Different basic methods are used to integrate these basic classifiers to obtain different performance DICNN variants. The performance of these methods are shown in Fig. 3.

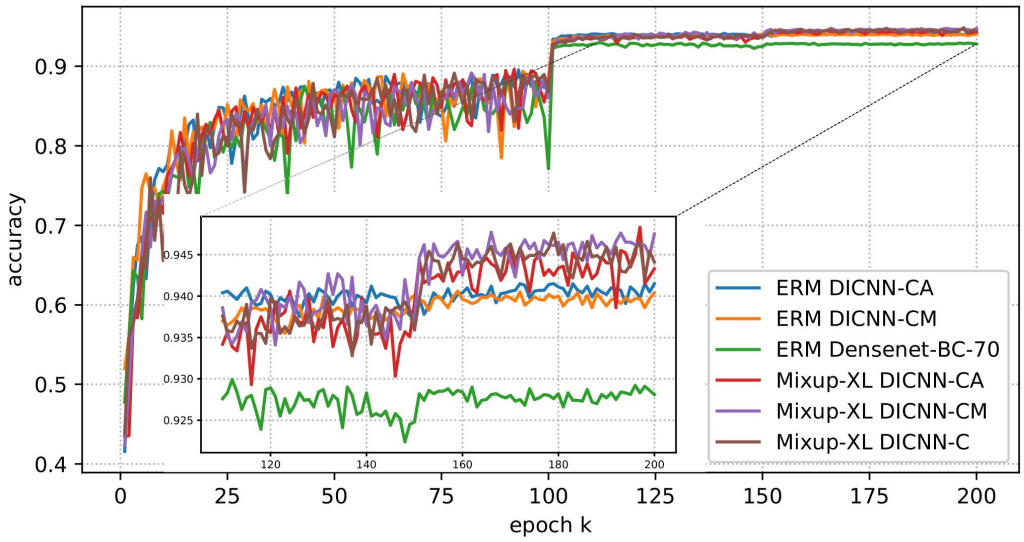


Fig. 3. Test accuracy of different network (DICNN and DenseNet-BC-70 [6]) structures and different training methods (ERM and Mixup-XL) on the CIFAR10 dataset. The subgraphs connected by dashed lines are the amplified test process after 110 epoch.

In Fig. 3, it can be seen that the performance of DICNN is slightly higher than that of DenseNet-BC-70 [6]. The performance of DICNN trained with Mixup-XL is higher than that of DICNN trained with ERM, which demonstrates the effectiveness of the DICNN architecture and the Mixup-XL training method. More specific performance tests on DICNN are shown in Table 3 and Table 4.

Table 3. DICNN performance using different training methods on CIFAR-10 [24].

	DenseNet-BC-70	DICNN-C	DICNN-A	DICNN-M	DICNN-V
ERM	93.1%	93.7%	93.4%	93.3%	93.5%
Mixup	93.8%	94.2%	94.1%	93.8%	94.1%
Mixup-XL	93.9%	94.5%	94.3%	94.2%	94.3%

Table 4. DICNN performance using different training methods on CIFAR-100 [24].

	DenseNet-BC-70	DICNN-C	DICNN-A	DICNN-M	DICNN-V
ERM	71.5%	72.8%	72.6%	72.5%	72.8%
Mixup	74.0%	74.5%	74.3%	74.3%	73.4%
Mixup-XL	75.2%	75.4%	75.4%	75.2%	75.4%

In Table 3 and Table 4, it can be found that DICNN-C combined with Mixup-XL can achieve the best performance. This is because DICNN-C uses a fully connected layer more than other methods in the final integration. By adding all the basic classifiers, DICNN-A also achieved relatively good performance. The performance of the DICNN-M is relatively lower, but almost consistently higher than the DenseNet-BC-70 [6]. This also illustrates the effectiveness of the DICNN architecture and the Mixup-XL method.

4.5 DICNN-V

In this section, we experimented with DICNN's voting algorithm, which uses the same network architecture as the Sect. 4.4. Eq. (7) is used as the loss function to train the networks with different C_{final}. Fig. 4(a) shows the test performance of DICNN which use Eq. (2) to obtain C_{final}. Since multiplying all the base classifiers directly to get C_{final} will cause the DICNN to not converge (using composite function f to do the regression can make it converge, just like Eq. (1)), we multiply any two base classifiers and add them as C_{final} of the DICNN, which is formalized as:

$$C_{final} = \sum_{i=1}^{N} \sum_{j>i}^{N} C_i C_j \tag{9}$$

Fig. 4(b) shows the test performance of training DICNN using Eq. (9) as C_{final}. In Fig. 4(a), all methods can be predicted normally. Compared to other methods, the performance of DICNN-M is slightly lower. This means that even if a classifier has the highest prediction probability for a category, it is possible to predict failure. DICNN-VM combines the advantages of DICNN-V and DICNN-M, and its test performance is the highest. This is because when some categories predict errors, it chooses the category with the highest probability as the final classifier. In Fig. 4(b), the voting algorithm based DICNN does not work properly. Using Eq (9) as C_{final} makes it impossible to vote with a single base classifier alone because they are multiplicatively related to each other. Somewhat strangely, DICNN-M still works, and its performance is similar to DICNN-A. The reason for this phenomenon is that there is competition between all basic classifiers when training DICNN using Eq (9). As a result, only some of the classifiers have larger predicted probability values, while other basic classifiers that fail in competition have less influence on the final prediction results.

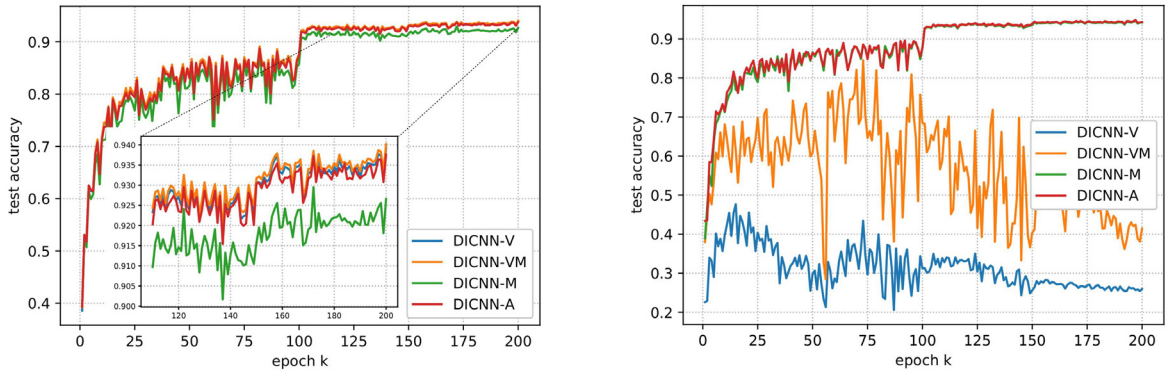


Fig. 4. Different voting algorithms are used to test DICNN

5 Conclusions

In this paper, a method similar to neural network integration is proposed. The method first divides an basic network architecture into multiple base classifiers in the depth direction and then integrates these base classifiers in a way similar to ensemble learning. Although deeply integrated convolutional neural network (DICNN) is a pseudo-ensemble learning method, it can achieve higher performance than the original network. In addition to training DICNN using traditional ERM method, a Mixup variant approach was proposed. This method uses different distributions to mix the samples and their labels, achieving better generalization performance than the original method. In the experimental section, the performance of various variants of DICNN were compared separately, and the adaptation based on the voting algorithm and its disadvantages were analyzed. This approach can be easily extended to other network architectures and can use a wider range of ensemble algorithms for better generalization performance.

Acknowledgements

The work is partially supported by the Technology and Development Project of Shandong (No.61373081) and the Taishan Scholar Project of Shandong, China.

References

[1] L. K. Hansen, P. Salamon, Neural network ensembles, IEEE Trans Pattern Analysis and Machine Intelligence 12 (10)(1990) 993-1001.

[2] P. Sollich, Learning with ensembles, How over-fitting can be useful. Advances in Neural Information Processing Systems 8(1996) 190-196.

[3] R. Maclin, D. Opitz, Popular ensemble methods: an empirical study, Journal of Artificial Intelligence Research 11(1999) 169-198.

[4] L.N. Cooper, Hybrid neural network architectures: equilibrium systems that pay attention, in: R.J. Mammone, Y.Y. Zeevi (Eds.), Neural Networks: Theory and Applications, Academic Press, 1991, pp. 81-96.

[5] K.-M. He, X.-Y. Zhang, S.-R. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[6] H. Gao, Z. Liu, K.Q. Weinberger, Densely connected convolutional networks, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[7] H. Zhang, M. Cisse, Y.N. Dauphin, D. Lopez-Paz, Mixup: beyond empirical risk minimization, in: Proc. International Conference on Representation Learning (ICLR), 2017.

[8] V.N. Vapnik, Statistical Learning Theory, John Wiley, 1998.

[9] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proc. International Conference on Neural Information Processing Systems (NIPS), 2012.

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z.-H. Huang, A. Karpathy, A. Khosla, M. Bernstein, Imagenet large scale visual recognition challenge, International Journal of Computer Vision 115(3)(2014) 211-252.

[11] C. Szegedy, W. Liu, Y.-Q. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[12] R.K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Proc. International Conference on Neural Information Processing Systems (NIPS), 2015.

[13] S. Xie, R. Girshick, P. DollÃar, Z.-W. Tu, K.-M. He, Aggregated residual transformations for deep neural networks, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[14] M.P. Perrone, L.N. Cooper, When networks disagree: ensemble methods for hybrid neural networks, in: Proc. Artificial Neural Networks for Speech and Vision, 1993.

[15] D. Opitz, J. Shavlik, Actively searching for an effective neural network ensemble, Connection Science 8(1996) 337-353.

[16] X. Zhang, J.P. Mesirov, D.L. Waltz, Hybrid system for protein secondary structure prediction, Journal of Molecular Biology 225(4)(1992) 1049-1063.

[17] R. Jabcobs, Adaptive mixtures of local experts, Neural Computation 232(2)(1991) 584-599.

[18] D. Jimenez, Dynamically weighted ensemble neural networks for classification, in: Proc. IEEE International Joint Conference on Neural Networks Proceedings, 1998.

[19] N. Ueda, Optimal linear combination of neural networks for improving classification performance, IEEE Trans Pattern Analysis and Machine Intelligence 22(2)(2000) 207-215.

[20] H. Inoue, Data augmentation by pairing samples for images classification. <https://arxiv.org/abs/1801.02929>, 2018.

[21] I. Sergey, S. Christian, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proc. International Conference on Machine Learning (ICML), 2015.

[22] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proc. International Conference on Artificial Intelligence and Statistics (AISTATS), 2011.

[23] K.-M. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: Proc. European Conference on Computer Vision (ECCV), 2016.

[24] A. Krizhevsky, G. E. Hinton, Learning multiple layers of features from tiny images, Tech Report, 2009.

[25] I. Sutskever, J. Martens, G. Dahl, G.E. Hinton, On the importance of initialization and momentum in deep learning, in: Proc. International Conference on Machine Learning (ICML), 2013.