华中科技大学
计算机科学与技术学院
School of Computer Science &Technology，HUST

COMPUTER VISION

Experimental Report 2

CS2110 U2021XXXXX Gao Lang
Huazhong University of Science and Technology

1 Introduction

1.1 Background

CNNs were introduced by Yann Lecun from New York University in 1998 (LeNet-5). At its core, CNN is an MLP that employs local connectivity and weight sharing. This approach reduces the number of weights, making the network easier to optimize. Additionally, it lowers the complexity of the model and minimizes the risk of overfitting. When applied to image inputs, these advantages become even more pronounced.Figure 1 shows the structure of a simple CNN.[1]

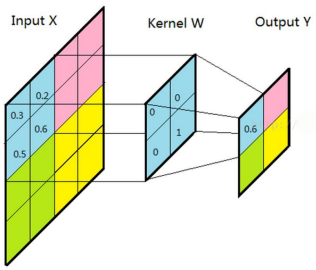

Figure 1:Basic structure of a CNN

Residual connection is a specific method for constructing deep learning models, proposed to address the degradation problem encountered during the training of deep models. Its structure is shown in Figure 2: Residual connections stack feature maps before and after several convolutions. The advantage of this approach is that it ensures that deeper models at least have the performance of shallower models, thereby avoiding the degradation problem in model performance.[2]

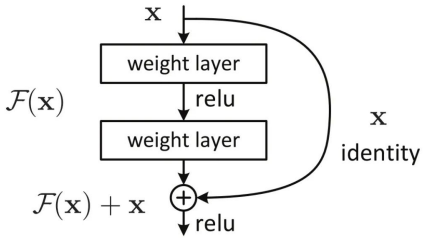

Figure 2:Structure of residual connection

1.2 Experimental Objective

MNIST is a dataset consisting of 60,000 training images and 10,000 testing images. Each image contains a label indicating the digit it represents. MNIST has been integrated into the *Keras* deep learning framework, making it rather easy to implement with just a few lines of code.Figure 3 is a snapshot of MNIST dataset.[3]

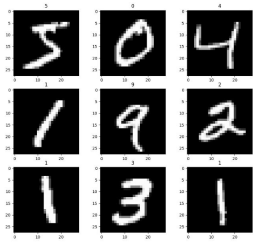

Figure 3:Snapshot of MNIST dataset

The objective of the experiments is to design a convolutional neural network and use residual connection module in it to achieve 10-class handwritten digit recognition on the MNIST dataset.

2 Framework

In the experiment, we designed the network input dimension as (28, 28, 1) and output dimension as (10) based on the size of the dataset images and the type of task. We defined the residual connection block, in which the input feature map is stacked on the feature map obtained after three convolutions (without pooling) and then output. Our model used several such residual connection blocks to construct two models of different depths, *resnet_shallow* and *resnet_deep*. The parameter scales of the models is shown in Table 1. In these two models, after passing through several residual connection blocks, the feature map will be flattened into a 1-dimensional vector and then passed through 2 fully connected layers to obtain the final 10-dimensional vector. We directly take the dimension with the largest value in the output vector as the result of the model's handwritten digit recognition classification.The Python implementation of the models resnet_shallow and resnet_deep can be found in **Appendix A**.

Table 1:Parameter scales of *resnet_shallow* and *resnet_deep*

MODEL NAME	BLOCK_NUM	TOTAL_PARAMS	TRAINABLE_PARAMS
resnet_shallow	2	98060(383.05KB)	97928(382.53KB)
resnet_deep	4	93740(366.17KB)	93032(363.41KB)

The sample labels in the MNIST dataset provided by *Keras* are numerals, which are not consistent with the form of the 10-dimensional vector obtained by the model's output. Therefore, we use the *to_categorical()* function to convert the sample labels into one-hot encoding. At the same time, we use *categorical_crossentropy* as the loss function and *accuracy* as the evaluation metric for the model's performance.

3 Experiments

3.1 Experimental setup

Datasets and Models In this experiment, we utilized the MNIST handwritten digit dataset, which is a built-in dataset in the Keras framework, as both the training set and testing set for our model. The training set consists of 60,000 samples, while the testing set contains 10,000 samples. We conducted a series of comparative experiments

using *resnet_shallow* and *resnet_deep* to investigate the differences in their capabilities in solving the same problem when equipped with residual connections of varying depths.

As shown in Figure 3,both resnet_shallow and *resnet_deep* are composed of residual connection modules and fully connected modules. They share the same structure except for the number of residual connection blocks used. Specifically, after passing through all the residual connection modules and pooling modules, they both flatten the feature maps into a one-dimensional form and input them into a fully connected layer with a dimension of 64. Subsequently, they connect a 10-dimensional fully connected layer with a softmax activation function as the output layer.

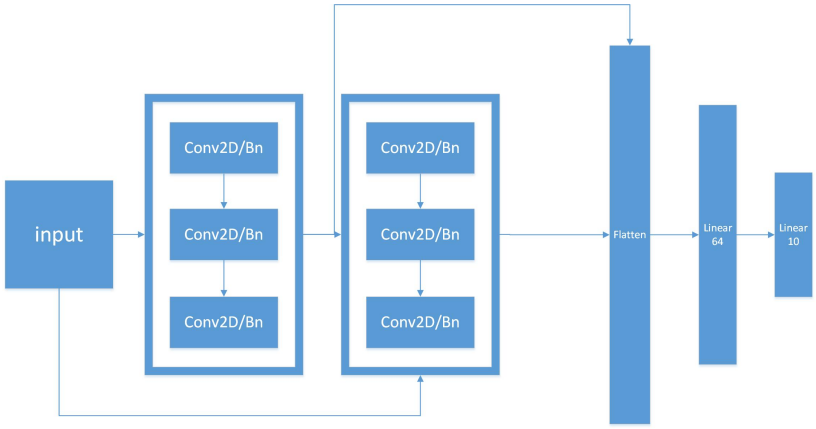

Figure 3:Basic structure of *resnet_shallow.*

Metrics We use Categorical cross_entropy as the loss function and plot the loss curve by calculating the loss for each batch on both the training set and testing set. We use accuracy(Acc) as the evaluation criterion for model performance and plot the curve of model accuracy over epochs.Besides,we also use precision(P),recall(R)and f1-score($F1$)to draw confusion matrix,in order to have a better understanding of the performance of models.

Implementation details During experiments, each model was trained for 1 epoch.All experiments in this report were completed on a single CPU.

3.2 Experimental Results

The content arrangement of this section is as follows: Firstly, the prediction effects obtained by using different model architectures and hyperparameters will be displayed and analyzed. Then, taking batch_size=64 as an example, the characteristics of loss and accuracy changes during training for both models will be compared, and it will be explained why using more residual connection blocks can significantly improve the model's ability. Finally, the impact of the dataset on model training will be analyzed in more detail by combining the confusion matrix.

3.2.1 Overall performance

Table 1 reflects the impact of changing the activation function and batch size of hidden layers on the prediction performance of different models.

Table 1:Experimental results on the dataset using different model hyper-parameters.The bolded data in each row represents the optimal model parameters.

Batch size	resnet_shallow				resnet_deep			
	Acc	*P*	*R*	*F*1	*Acc*	*P*	*R*	*F*1
64	0.93	0.93	0.93	0.93	**0.96**	**0.96**	**0.96**	**0.96**
128	0.91	0.91	0.91	0.91	**0.94**	**0.94**	**0.94**	**0.94**

From Table 1, it can be concluded that the model's feature extraction ability is stronger and its performance is better when more residual connection modules are used under the same hyperparameter conditions. At the same time, we can also conclude that using a larger batch_size does not necessarily lead to better classification results under the same model architecture. In fact, it can be concluded that using a larger batch_size may have a negative impact on model training in the ten-classification task of handwritten digits.

3.2.2 Training process comparison

Below, taking *batch_size=64* as an example, the differences in loss and accuracy trends between *resnet_shallow* and *resnet_deep* during training on both the training set and test set will be analyzed. Furthermore, it will be demonstrated that *resnet_deep* has superior performance.We also conducted the same test under the condition of parameter *batch_size=128*.Please refer to **Appendix B** for more details.

Figure 4 illustrates the changes in four metrics of the model during training: loss, accuracy, validation loss, and validation accuracy. The *x* of the 4 sub-graphs in Figure 4 are all the number of times the model parameters are updated. Due to the problem of slow testing speed caused by the large size of the test set, after each parameter update, we randomly select *batch_size* samples from the test set as the validation set for this update, and only calculate the loss and accuracy of the model on these samples, without participating in updating the model's parameters. Since the large loss in the early stages of training leads to relatively insignificant changes in later stages, the losses in the first 20 updates were excluded when plotting the loss graph.

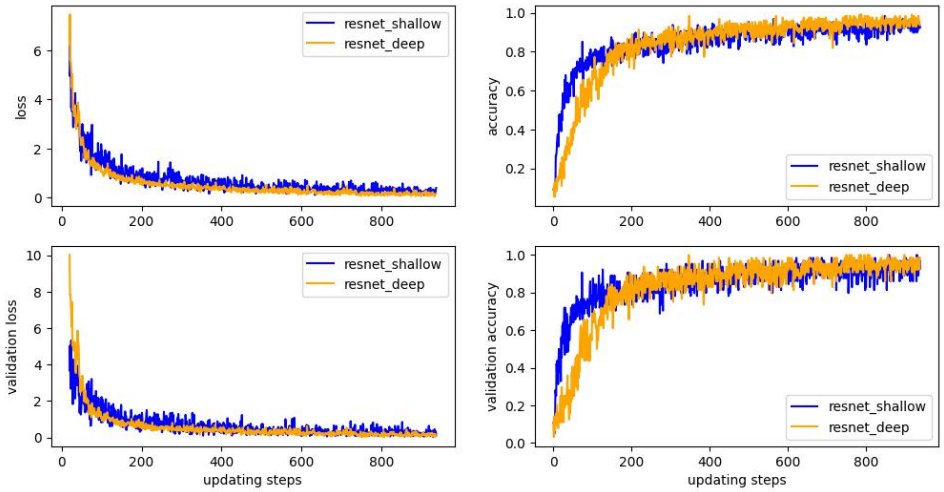

Figure 4:Loss,validation loss,accuracy,validation accuracy of the models during training.

Table 1 shows that *resnet_shallow* has 98,060 parameters(fp16), which is 4,320 more than the 93,740 parameters of

resnet_deep. However, as can be seen from Figure 4, its convergence speed is faster than that of *resnet_deep*. In contrast, *resnet_deep* almost always had higher loss in the first 100 round of parameter updates and consistently lower accuracy on both the training set and validation set in the first 200 round of parameter updates compared to *resnet_shallow*. However, in the later stages of training, *resnet_deep* was able to achieve lower loss values for most of the time compared to *resnet_shallow*, while also achieving higher prediction accuracy on both the training set and validation set.Overall, *resnet_deep* has better performance than *resnet_shallow* in the handwritten digit recognition task.

3.3.3 Performance on different samples

Figure 5 presents the performance of the two models on different samples of the entire test set in the form of a confusion matrix. It can be observed that the f1 scores obtained from the predictions of different category models are also different. Overall, the prediction results for categories 1, 2, 3, 4, 5, and 6 are satisfactory, but those for categories 7, 8, and 9 are slightly worse. This is not due to an improper distribution of training data. Figure 6 shows the distribution of samples from the MNIST training set across 10 categories, with a relatively balanced proportion, which can rule out the influence of uneven distribution.Please note that these experiments were conducted under the condition of batch_size=64. For the results of batch_size=128, please refer to **Appendix C**.

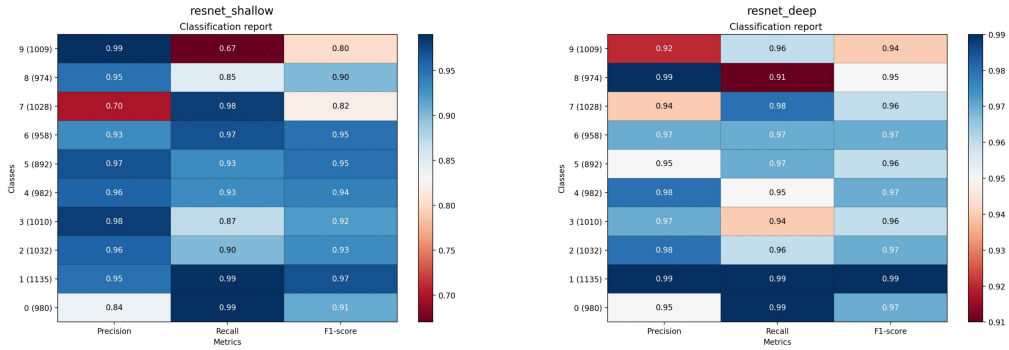

Figure 5:Confusion matrix of *resnet_shallow* and *resnet_deep*.The models perform differently on different kinds of samples.

In addition, we can also find that the prediction precision for category 7 is very low in both models, category 8 has a lower recall value, and category 9 has a precision of 0.99 in resnet_shallow and a precision of 0.92 in resnet_deep. These phenomena tell us that handwritten digit recognition task is not a whole, and the difficulty of recognizing and classifying different digits varies for models. The problems behind these phenomena will be explained in detail in the discussion section.

4 Discussion

In the following parts, we will explore some issues arising from the experimental data.

Why model with more parameters converges faster and performs not good enough instead? The experimental results in Figure 4 reveal two important issues: during the training phase, the convergence speed of the model's loss and accuracy is not entirely related to the number of parameters, and the lower limit of the model's loss and the upper limit of its accuracy are not entirely related to the number of parameters as well. *resnet_shallow* has more parameters than *resnet_deep* and should take longer to converge, with better performance, but in reality, it

converges faster and has weaker classification ability than *resnet_deep*. Table 2 shows the parameter distribution of *resnet_shallow* and *resnet_deep*, from which it is clear that most of the parameters of *resnet_shallow* are concentrated in the first linear layer. This layer's function is to perform the final feature aggregation on the feature maps output by the residual connection blocks and pass them to the output layer. It does not need to re-extract information based on the original image, so its importance and complexity of the task it needs to complete are both inferior to those of the residual connection blocks. Therefore, *resnet_shallow* is a simpler model structure. This illustrates that ultimately, the difficulty and representational ability of model training are actually positively correlated with the complexity of the model architecture.

Table 2:Parameter distribution of *resnet_shallow* and *resnet_deep*

Model NAME	residual blocks' params	linear layers' params	total params
resnet_shallow	3138(3.33%)	94922(96.80%)	98060
resnet_deep	62562(67.25%)	31178(33.26%)	93740

Why do models have different abilities to predict different numbers? This may be related to the difficulty of recognizing the numbers themselves Number 1 has achieved the highest f1 score in both models, and with the increase in model complexity, the f1 score does not decrease, which indicates that some numbers (such as 1) have distinctive features, their features are not the same as those of other numbers, and they are not misleading, making it possible for any size of model to correctly recognize them. The fact that precision for number 9 drops from 0.99 in a simple model to 0.92 in a complex model suggests that number 9 also has distinctive features, but these features overlap with those of other numbers to some extent, leading to overfitting when using a complex model for training due to excessive recognition of features. On the other hand, both models perform poorly on number 7. This may be related to the different writing habits of number 7. Figure 7 shows some samples of number 7 and 9, and it can be observed that even human eyes find it difficult to correctly identify all instances of number 7. Number 9 is similar to number 7 in some cases, which further validates the hypothesis of overfitting.

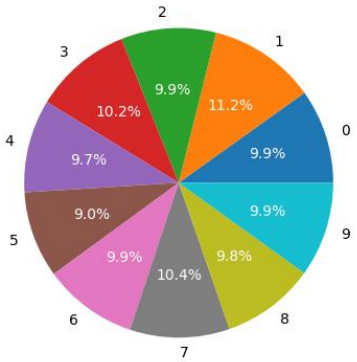

Figure 6:Proportions of Numbers in MNIST.

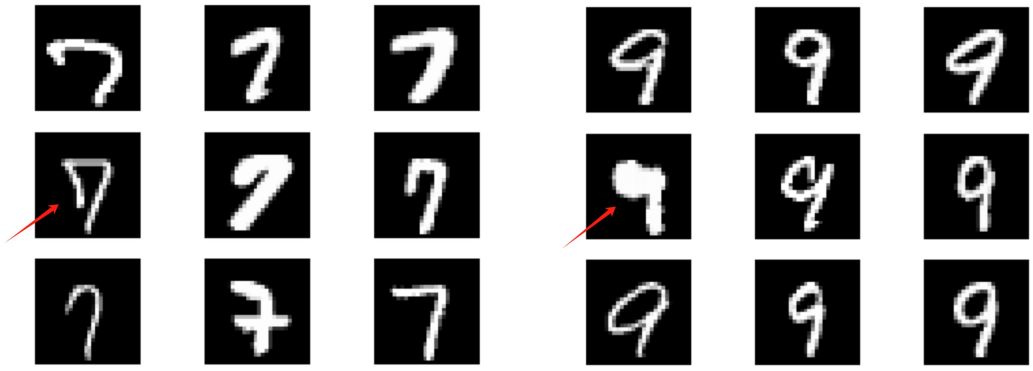

Figure 7:Some samples of numbers 7 and 9 in the MNIST test set. The red arrows point to samples that are prone to confusion, with the ones on the left being mistaken for 9 and those on the right being mistaken for 7.

5 Conclusion

In this experiment, we designed convolutional neural networks with residual connections to solve the problem of MNIST handwritten digit recognition. We designed *resnet_shallow* with fewer residual connection modules and *resnet_deep* with more residual connection modules to complete this task together, and also tested the training results of these two models under the hyperparameters of *batch_size=64* and *batch_size=128*. We found that even though the model with more residual connection modules had slightly fewer total parameters than the simple model, its convergence speed was still slower and the training accuracy still reached a higher level. This may essentially illustrate that the convergence speed and fitting ability of the model are not directly positively correlated with the number of parameters, but are more likely positively correlated with the complexity of the model structure itself. We also explored the impact of the dataset on the model's ability, and ultimately concluded that different numbers have different difficulty levels for recognition tasks. At the same time, some numbers often contain misleading information due to different writing habits, which makes it more difficult for the model to judge, leading to confusion. Finally, in this task, larger batch sizes do not bring better results, which may be related to only 1 epoch of training and data volume, so there is a trade-off between training accuracy and training speed.

Reference

[1] 【Overview】Understanding Convolutional Neural Networks (CNN) in One Article [Blog post]. Retrieved from https://zhuanlan.zhihu.com/p/561991816

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[3] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Appendix A

Implementation of *residual_block,resnet_shallow* and *resnet_deep* using *Keras* structure.

```
def residual_blocks(num_filter,input):
    conv1 = Conv2D(filters=num_filter, kernel_size=(2, 2), strides=1, padding='same')(input)
    bn1 = BatchNormalization(axis=-1)(conv1)
    conv2 = Conv2D(filters=num_filter, kernel_size=(2, 2), strides=1, padding='same')(bn1)
    bn2 = BatchNormalization(axis=-1)(conv2)
    conv3 = Conv2D(filters=num_filter, kernel_size=(2, 2), strides=1, padding='same')(bn2)
    bn3 = BatchNormalization(axis=-1)(conv3)
    res=concatenate([input,bn3],axis=-1)
    return res

def resnet_shallow(input_shape=(28,28,1),classes=10):
    X_input = Input(input_shape)
    #0 padding
    X = ZeroPadding2D((2,2))(X_input)
    res1=residual_blocks(num_filter=6,input=X)
```

```
    pool2=MaxPooling2D(strides=2)(res1)
    res2=residual_blocks(num_filter=16,input=pool2)
    pool3=MaxPool2D(strides=2)(res2)
    linear=Flatten()(pool3)
    dense1=Dense(64,activation='relu')(linear)
    out=Dense(classes,activation='softmax')(dense1)
    model=keras.Model(inputs=X_input,outputs=out,name='resnet_small')
    return model

def resnet_deep(input_shape=(28,28,1),classes=10):
    X_input = Input(input_shape)
    #0 padding
    X = ZeroPadding2D((2,2))(X_input)
    res1=residual_blocks(num_filter=6,input=X)
    pool2=MaxPooling2D(strides=2)(res1)
    res2=residual_blocks(num_filter=16,input=pool2)
    pool3=MaxPooling2D(strides=2)(res2)
    res3=residual_blocks(num_filter=32,input=pool3)
    pool4=MaxPooling2D(strides=2)(res3)
    res4=residual_blocks(num_filter=64,input=pool4)
    pool5=MaxPool2D(strides=2)(res4)
    linear=Flatten()(pool5)
    dense1=Dense(64,activation='relu')(linear)
    out=Dense(classes,activation='softmax')(dense1)
    model=keras.Model(inputs=X_input,outputs=out,name='resnet_large')
    return model
```

Appendix B

Figure 8 shows the changes in various indicators of model training under the condition of *batch_size=128*.The lag in convergence and superiority in training accuracy in the later stages of training are more pronounced in complex models compared to *batch_size=64*.

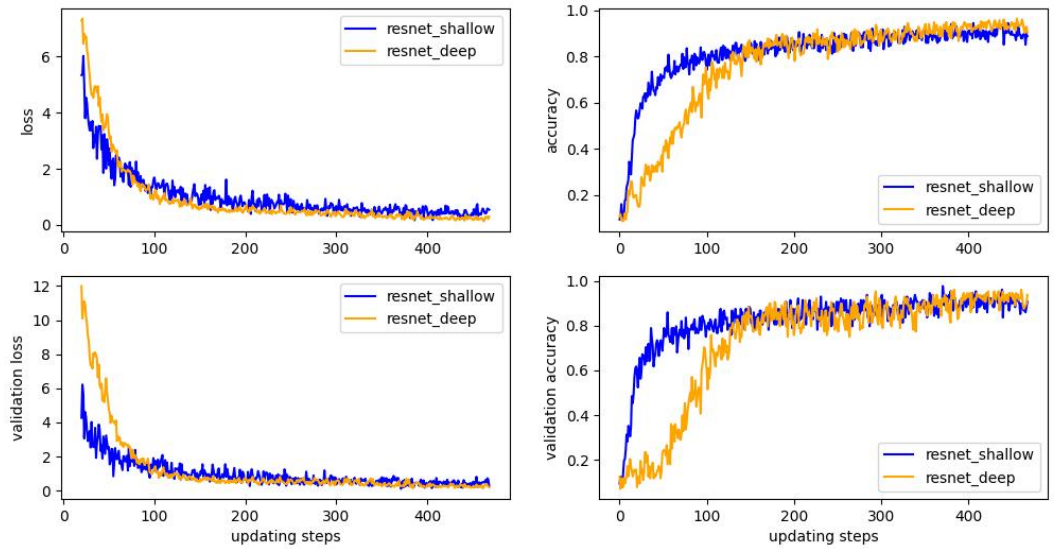

Figure 8: Loss,validation loss,accuracy,validation accuracy of the models during training with *batch_size=128*.

Appendix C

Figure 9 presents the various indicators of the model's predictions on different samples under *batch_size=128* in the form of a confusion matrix.Complex models can achieve better training results on number 7. However, in the prediction of numbers 8 and 9, it can be observed that overfitting occurs when using complex models for training. Additionally, number 2 was also identified as having misleading information.

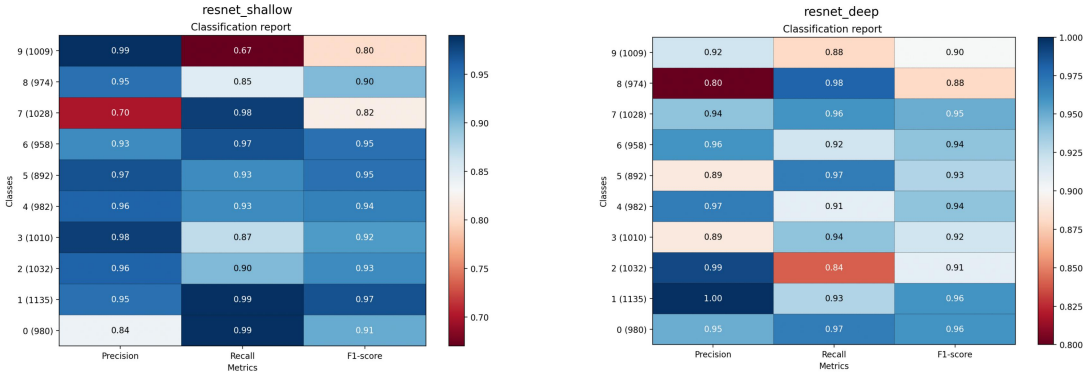

Figure 9:Confusion matrix of *resnet_shallow* and *resnet_deep*.The models perform differently on different kinds of samples with *batch_size=128*.