COMPUTER VISION

Experimental Report 3

CS2110 U2021XXXXX Gao Lang
Huazhong University of Science and Technology

1 Introduction

MNIST is a dataset for handwritten digit recognition, integrated into the deep learning framework Keras. The MNIST dataset contains 70,000 (28,28) handwritten digit images, of which 60,000 are divided into training sets and 10,000 are divided into test sets. This experiment is based on the MNIST dataset and requires inputting two (28,28) handwritten digital images, and the model should be able to detect whether the two images represent the same number. The experiment requires selecting 10% of the MNIST training set as the training images for this experiment, and selecting 10% of the MNIST test set as the test images.

2 Framework

In this experiment, we use a convolutional neural network (CNN) with residual connection blocks as the main module for feature extraction in Experiment 2. We also design two classification heads: one for the task of classifying handwritten digits as different classes, and another for the task of comparing handwritten digits as same or different classes.
We adopt the idea of transfer learning: since the task of judging whether the two numbers are the same has a great similarity with the number recognition task, logically speaking, if we first train a traditional model for handwritten digit recognition, it will help the model retain the standardized writing features of ten digits and reduce the challenge of distinguishing between two images when given two pictures. After the traditional handwritten digit recognition model is trained, we freeze all parameters of the feature extraction module and replace the classification head with the binary classification head for the task of comparing handwritten digits as same or different classes. We then train this classification head on the dataset of comparing handwritten digits.The parameter scales of each module in the model are shown in Table 1.

Table 1 Parameter scales of modules in the model

Module Name	Output Shape	Total Parameters
Feature_Extraction	(batch,1536)	3226
10_head_classifier	(batch,10)	99018
2_head_classifier	(batch,2)	401762

The specific structure of the model is shown in figure1.The implementation using *Keras* can be found in **Appendix A.**

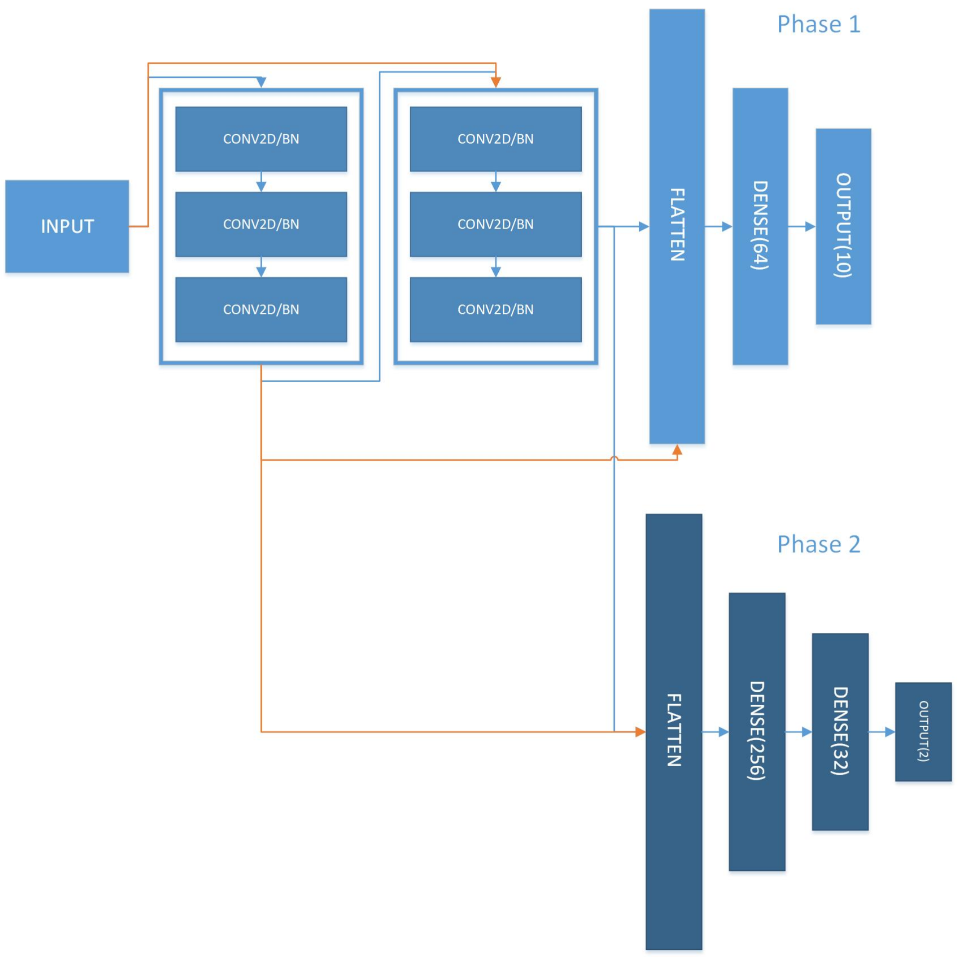

Figure 1:The structure of the model.

We have 2 training phases where the Feature_Extraction module is combined with different classifier.In Phase 1 we simply train a 10-class classifier using the original MNIST dataset,and in Phase 2 we use the 2-class classifier to train on our customized dataset.The orange arrows are residual connections.

3 Experiments

3.1 Experimental setup

Datasets In this experiment, we used the MNIST dataset built-into the deep learning framework Keras. The training set consists of a total of 60,000 samples, while the testing set consists of 10,000 samples. According to the experimental requirements, we randomly constructed 6,000 image pairs as the original data for the training set and 1,000 data points as the original data for the testing set. Then, we randomly selected 100,000 unique image pairs from the 6000 training set and 10,000 unique image pairs from the 1000 testing set. To ensure that the dataset does not have any biases, we added certain constraints to the random construction: the probability of selecting two images representing the same number should be 50%. The process of constructing the dataset is roughly shown in Figure 2.

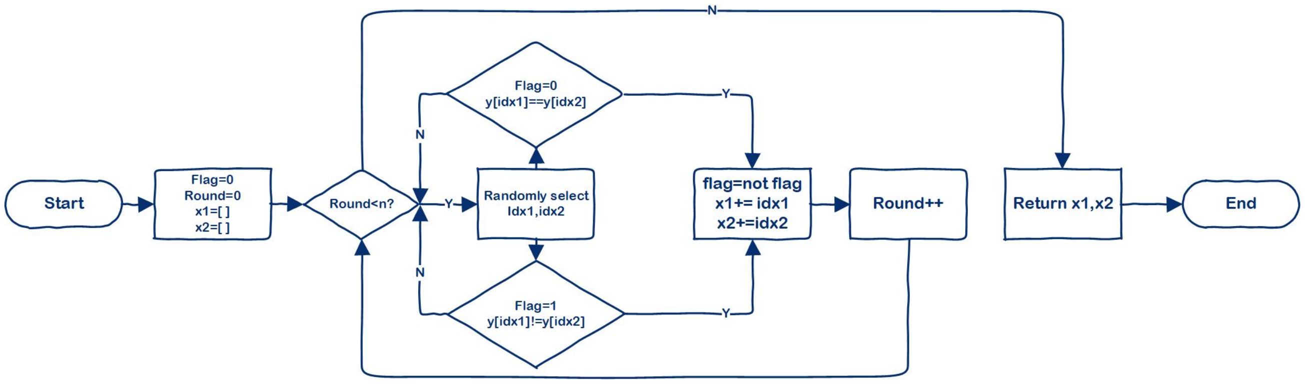

Figure 2:Process of generating training data and testing data.

After selecting the indices of the parade data and test data, we use the **Numpy** library to slice and stack the original dataset into a format suitable for model input. Specifically, for the training dataset with a shape of (60000,28,28), we use the values at corresponding positions in the two index lists to obtain the corresponding (1,28,28) matrix slices as inputs. Then we call the np.stack function to stack the two images obtained at the last dimension, forming a sample of the number comparison dataset with a shape of (1,28,28,1). Finally, we stack all samples in the first dimension to form the final training data with a dimension of (6000,28,28,1).
Next, we use logical operations on matrices to obtain the label y indicating whether the two images represent the same number. To match the format of the model output as a two-dimensional vector, we denote [0,1] as representing different numbers and [1,0] as representing the same number. The final label shape of the number comparison dataset is (6000,2). We perform the same processing for the test set.Please refer to **Appendix B** for Numpy implementation on building datasets.

Models In terms of the model architecture, we use a similar structure as in Experiment 2. For the Feature_Extraction module of the model, we use a structure where three residual connection blocks are stacked with max pooling layers. Each residual connection block contains three 2D convolutional layers and three batch normalizations, and its output is the stacking of the input and the computation result of the last layer in the last dimension. The classification head for the 10-classification task includes a flattening layer, a linear layer with an output dimension of 64, and an output layer with an output dimension of 10; the classification head for the binary-classification task includes a flattening layer, two linear layers with output dimensions of 256 and 32 respectively, and an output layer with an output dimension of 2. In both training phases, we use batch_size=64 as a hyperparameter.

Metrics We use Adam optimizer during training stage.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 classification report,in order to have a better understanding of the performance of model.

Implementation details During experiments, in each training phase the model was traine for 2 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

After training the model, it was tested on a test set of 10,000 images. The overall results were: P=0.89, R=0.89, $F1$=0.89, and Acc=0.89. More specifically, Figure 3 shows the test performance of the model on two categories.

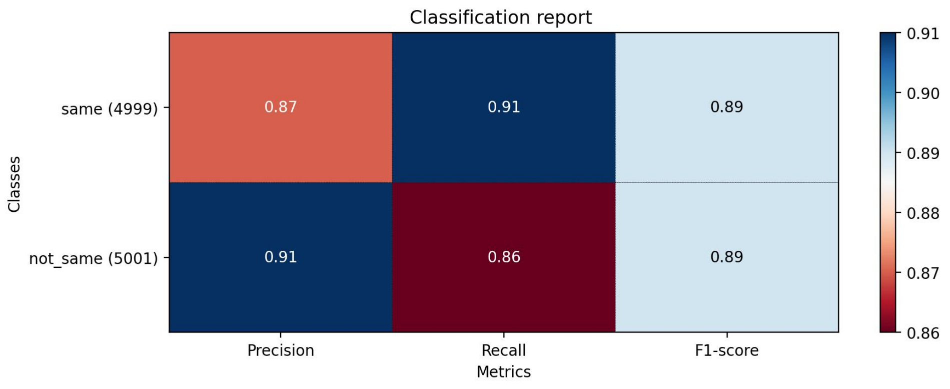

Figure 3:Test performance on two categories.

As can be seen from the figure, the F1 scores obtained by the model for the two categories are equal, indicating balanced overall performance. However, the precision of the same category is lower while the recall is higher, and the precision of the not_same category is higher while the recall is lower. This suggests that the model tends to classify two input images as different numbers, but there is a high probability that samples classified as the same category actually represent the same number. We believe that the model is able to effectively utilize the feature extraction method obtained in phase 1.

3.2.2 Training process comparison

After each parameter update, we tested the model on a small validation set, which was obtained by randomly selecting batch_size samples from the test set. The loss calculated on the validation set was not used in backpropagation. Figure 4 shows the loss and accuracy of the model on the training set and validation set during training in phase 2.

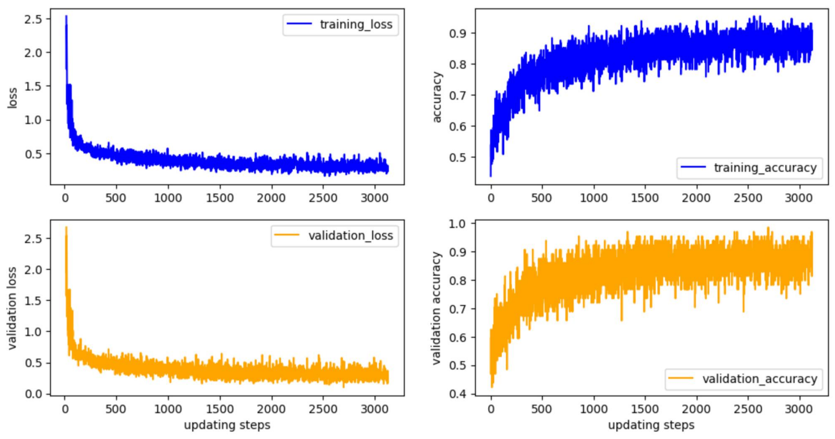

Figure 4:Loss and accuracy on training set and testing set.

As can be seen from the figure, during training, the model successfully achieved gradient descent and an increase in accuracy. However, in the middle and later stages of training, the decline in model loss slowed down and the fluctuations became larger. The increase in accuracy on both the training set and test set also slowed down and became more volatile, and the model was unable to stabilize above 0.85. This also indicates that there are certain flaws in the current model and data that need to be addressed.

4 Discussion

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

Why does the model tend to judge that the corresponding numbers of the images are different? This may be related to CNN feature extraction. Figure 5 is a schematic diagram of the convolution operation performed by CNN to generate feature maps. When the input data is highly similar and the parameters remain unchanged, the resulting feature map from convolution is also similar. If two images represent the same number, intuitively, the overlap rate between the pixels of the two images is high and if they are added in the last dimension, it is still possible to distinguish the represented number. Therefore samples of the same class are more likely to mainly include patterns of actual numbers 0-9, and the feature maps generated by convolution have high similarity and significant features. However, not_same samples can be any combination of two numbers, forming complex and irregular feature maps. The model will automatically learn such complex features instead of ignoring them. Therefore, in most of the feature maps of various modes, most of them are not_same types, so the model tends to be classified into not_same categories. Another possible reason is that when the model was trained for ten classifications in phase 1, it did not achieve a very high classification accuracy (about 93%), so it is difficult to further improve the classification accuracy of same class samples under frozen parameters (the current recall has reached 91%).

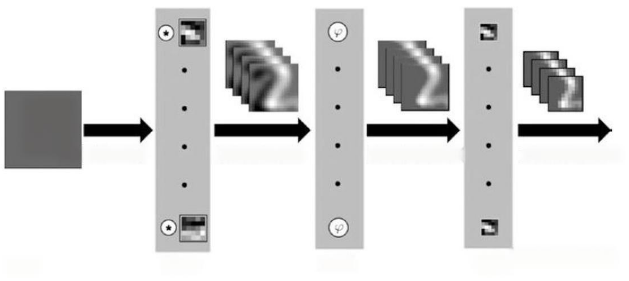
Figure 5: Convolution operation of CNN.

Why are there large fluctuations in loss and accuracy during the later stages of model training? One possible reason is the setting of model parameters. In this case, we directly froze the parameters of the feature extraction part of the model. Although both ten-classification and binary-classification tasks require identifying numbers first, they cannot fully share the same parameters. For example, during the training process in phase 1, in order to cater to the input with 2 channels of features, we copied each sample from the original MNIST dataset twice and stacked them together. This results in two identical inputs, causing the model to not focus on situations where the two images are different. Therefore, the feature extraction strategy actually needs to be changed. A specific approach could be: during phase 2 training, the feature extraction part of the model is trained with a different learning rate and update frequency than the classification part. Another possible reason is that the model structure is not set up properly. The current model is limited by the size of the input data, making it difficult for its feature maps to have a larger size. This makes it challenging to design a wider model, while an excessively deep model is prone to overfitting and unable to focus on more extensive feature information. Based on this, we can perform certain data processing operations, such as using bilinear interpolation or other algorithms to expand its size, allowing the model to obtain larger feature maps.

5 Conclusion

In this experiment, we used a convolutional neural network with residual connection blocks to complete the task of comparing the corresponding numbers of any two samples in the MNIST dataset. We first constructed a dataset suitable for this task, and then designed the structure of the network. We adopted the idea of transfer learning, first training an MNIST digit ten-classification model, and then connecting the feature extraction module with an untrained binary classifier for training the binary classification task of the dataset for this task, during which we gained insight into the parameters of the feature extraction module. In the end, we achieved a model performance close to 0.9. At the same time, we also found that there were certain problems with our feature extraction method and model architecture, and later needed to improve the training method of the feature extraction module, as well as expand the size of the dataset to build a larger model.

Appendix A

Implementation of the CNN model 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
    # return bn3

def feature_extraction(input_shape=(28,28,2)):
    X_input = Input(input_shape)
    #0 填充
    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)
    out=Flatten()(pool3)
    model=keras.Model(inputs=X_input,outputs=out,name='feature_extraction')
    return model
def classifier_10(input_shape=(1536,)):
    X_input=Input(input_shape)
    dense1=Dense(64,activation='relu')(X_input)
    out=Dense(10,activation='softmax')(dense1)
    model=keras.Model(inputs=X_input,outputs=out,name='10_head_classifier')
    return model
```

```
def classifier_2(input_shape=(1536,)):
    X_input=Input(input_shape)
    dense1=Dense(256,activation='relu')(X_input)
    dense2=Dense(32,activation='relu')(dense1)
    out=Dense(2,activation='softmax')(dense2)
    model=keras.Model(inputs=X_input,outputs=out,name='2_head_classifier')
    return model
Feature_Extraction=feature_extraction()
Cls_10=classifier_10()
Cls_2=classifier_2()
model_10=keras.Sequential()
model_10.add(Feature_Extraction)
model_10.add(Cls_10)
from keras.models import load_model
model_2=keras.Sequential()
Pretrained_Feature_Extraction=load_model(r'.\feature_extraction.h5',
                                       custom_objects=None,
                                       compile=True,
                                       options=None,
                                       )#报错是因为绝对路径中有中文名称
model_2.add(Pretrained_Feature_Extraction)
model_2.add(Cls_2)
model_2.layers[0].trainable=False#freeze params
model_2.layers[1].trainable=True
model_2.summary()
```

Appendix B

Numpy implementation on building datasets.

```
def generate_index(n, range_start, range_end,y):
    flag=0
    if n > (range_end - range_start + 1) ** 2:
        raise ValueError("no enough indexs")

    unique_numbers = set()
    while len(unique_numbers) < n:
        num1 = random.randint(range_start, range_end)
        num2 = random.randint(range_start, range_end)
        if flag==0 and y[num1]==y[num2]:
            continue
        elif flag==1 and y[num1]!=y[num2]:
            continue
        unique_numbers.add((num1, num2))
```

```
            flag=not flag
    x1,x2=[list(t) for t in zip(*unique_numbers)]
    return x1,x2
def make_dataset(x_train,y_train,x_test,y_test):
    x1_train_idx,x2_train_idx = generate_index(100000, 0, 6000,y_train[:6000])
    x1_test_idx,x2_test_idx=generate_index(10000,0,1000,y_test[:1000])

    x1_bi_train=np.stack(np.take(x_train,x1_train_idx,axis=0),axis=0)
    x2_bi_train=np.stack(np.take(x_train,x2_train_idx,axis=0),axis=0)
    x_bi_train=np.stack((x1_bi_train,x2_bi_train),axis=-1)
    y1_bi_train=np.stack(np.take(y_train,x1_train_idx,axis=0),axis=0)
    y2_bi_train=np.stack(np.take(y_train,x2_train_idx,axis=0),axis=0)
    y_bi_train=np.array([[0,1] if (a == b) else [1,0] for a, b in zip(y1_bi_train,
y2_bi_train)])

    x1_bi_test=np.stack(np.take(x_test,x1_test_idx,axis=0),axis=0)
    x2_bi_test=np.stack(np.take(x_test,x2_test_idx,axis=0),axis=0)
    x_bi_test=np.stack((x1_bi_test,x2_bi_test),axis=-1)
    y1_bi_test=np.stack(np.take(y_test,x1_test_idx,axis=0),axis=0)
    y2_bi_test=np.stack(np.take(y_test,x2_test_idx,axis=0),axis=0)
    y_bi_test=np.array([[0,1] if (a == b) else [1,0] for a, b in zip(y1_bi_test,
y2_bi_test)])
    return (x_bi_train,y_bi_train),(x_bi_test,y_bi_test)
```