Handwritten Chinese Sentence Recognition and Classification Based on Deep Convolutional Neural Networks

Chenyu Wang, Ganquan Wen, Jiali Ge, Runzhou Han, Yuchen Wang
wangcy@bu.edu, wengq@bu.edu, ivydany@bu.edu, rzhan@bu.edu, wangyc95@bu.edu
GitHub: https://github.com/GanquanWen/BU-EC500-Deep-Learing-Project

Figure 1. Handwritten Chinese characters

1. Task

In this project, we build own own model to give semantic classification for a handwritten Chineses sentences using deep convolutional neural network (CNN). We also evaluated the performance of our model and gave analysis about the performance. The work can be divided into three main parts: sentence segmentation, character recognition and sentence classification

2. Related Work

Handwritten digits recognition is one of the popular research fields of NLP. Amounts of groups have contributed to the topic.

Handwritten Digits Recognition Base on Improved LeNet5[1] mainly combine CNN and SVM together to realize the handwritten digits recognition. The paper replaced the last two layers of the LeNet5 structure with SVM classifier. The main point here is: Apply CNN to extract feature vectors and use SVM to classify them. In addition, the paper used stochastic diagonal Levenberg-Marquardt to accelerate the training process of CNN. The fault rate of the new method is 0.85% which is lower than that of CNN and SVM. The potential flaw of the method is LetNet5 has more feature maps, a large fully-connected layer and has a distributed representation to encode the categories at the output layers, rather than the more traditional “1” of “N” code.

A very high accuracy handwritten character recognition system for Farsi/Arabic digits[2] combined a number of Convolutional Neural Networks with gradient descent training algorithm. It is concentrated on two main contributions. The first one is automating extraction of input pattern features by using a CNN for Farsi digits and the second one is fusing the results of boosted classifiers to compensate the recognizers' errors. It used IFH-CDB database and applied two preprocess method to deal with the training result of the approved network. The potential flaw of the method is that it is quite similar to that of the MNIST network model. So it inherits the potential flaw of the MNIST network model.

Convolutional Neural Network Committees For Handwritten Character Classification[3] applied a group of convolution neural network which means 7 GPU train different datasets simultaneously. Then classify due to the output of 7networks. The false rate of the new method is clearly lower than the result of a single neural network. The potential flaw of the method is that the calculation needs a long period of time and it has a high demand for GPU.

3. Approach

Step I: Image segmentation

We need to segment sentence images into single characters for the next step of character recognition. We do the segmentation by the following steps:

1. Grayscale: Use OpenCV to read the image and change it to gray. The scale of gray is from 0 (black) to 255(white). We use 200 as the threshold to binarize the image, under 199 are black dots and over 200 are white dots. We have tried different values. 200 is good for the dataset we are using. Because the dataset images have clean and bright background and the color of the chinese characters are light.
2. Split the image into lines: We scan the image row by row (vertically) and count the amount of dots for each row. We can set a threshold to eliminate the effect of noise or sentence

overlapping. After scanning we can see the boundary of each sentence. For example, figure 2 is the original image that consists of two lines. In this example, the first line is located between break point 1 and break point 2. The second line is between break point 2 and break point 3. Figure 3 is the vertical scanning of it. There also exists images with overlap between lines, like the example shown in figure 4.

This part is optional because all our dataset images only have one line of sentence.

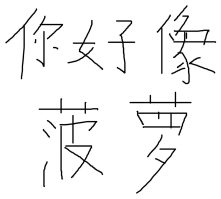

Figure 2. Original image consists of two lines

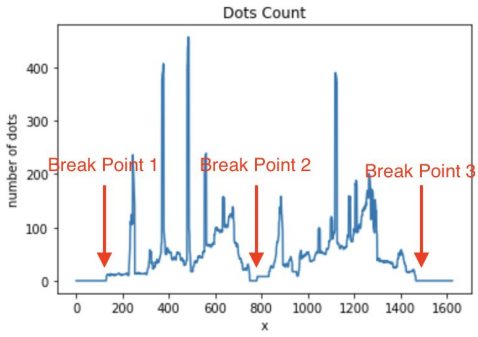

Figure 3. Vertical scanning of example in figure 2

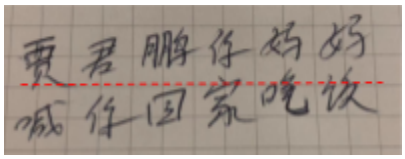

Figure 4. Example of Overlapped Sentence

3. Split each sentences into characters: We scan the sentence column by column (horizontally) and count the amount of dots for each column. It is similar to previous step. First we set a threshold value to find the border of each character. It is for eliminating the effect of character overlapping. Figure 5 shows the example of overlapped characters.

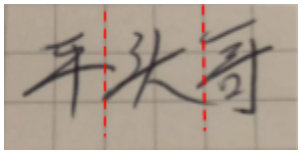

Figure 5. example of overlapped characters

Another problem is that some chinese characters contain multiple parts (mostly two). Figure 6 shows the situation of false border. These kind of characters can be seen as multiple characters. We need to find out these false border and ignore them.

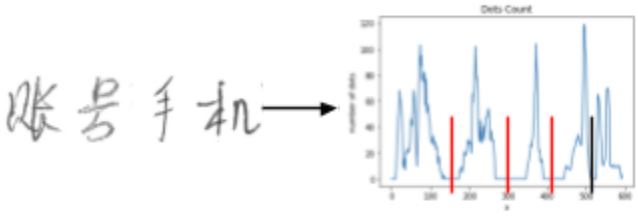

Figure 6. Example of false border(black line in the plot)

In this situation, we have two methods:

1. All the chinese characters are basically a square, we can estimate the size of a single character according to the height of a single sentence. We set the minimum size of a single segment to 70% of the height of the sentence. It performed well on our dataset.
2. We keep scanning for a few more times after the break point is detected. But we didn't implement it in our final version. It did solves this problem well. But it caused another problem. It judged two detached characters as one because they were too close. The program would make mistake when the gap between two characters is smaller than the gap inside the character with multiple parts.

An output of sentence segmentation looks like figure 7.

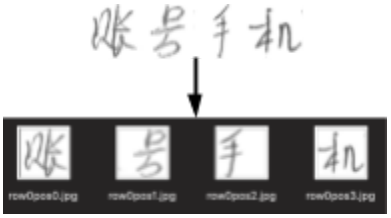

Figure 7. Example of sentence segmentation output

Step II: Character recognition

Given a RGB or grayscale image of a character, we use a convolutional neural network to recognize (classify) it. We constructed our own convolutional neural network which support 3755 class of characters with Tensorflow The configuration from Yuhao Zhang's work was used as a reference [4].

The libraries involved are Tensorflow and Numpy. We initialized our variables with tensorflow built-in function

global variable initializer. The loss function is cross entropy, and we set Relu to be our activation. Adam is used to be the optimizer.

Training a CNN with a large number of class can be difficult. It heavily relies on a wise choice of every parameter. Following are the experiment we did for our final design:

1. Batch size: We shuffled the dataset in every epoch and set the batch size to be 128, 256 and 512. We tested our model after every 10 training batch. From the test accuracy we found 256 image per batch converged much faster than 128, and varied little from 512.
2. Number of layers: We tried number of convolutional layers from 4 to 7. It result showed that a deeper model sometimes might not be the best choice. Before the layer number reached 6, the improvement in accuracy was obvious. However, when the layer number exceeded 6, it showed a decline both in efficiency and accuracy. Our final choice of depth was 6 convolutional layers followed by pooling layers, and 2 fully-connected layers in the end.
3. Other parameters: Number of filters, kernel size, number of units in the second last fully-connected layer. It didn't show obvious improvement between different choice of these parameters. However, with the model getting deeper, the loss was not decreasing as well as our expectation, so we thought about the effect of learning rate. It seems that a deeper model is usually more sensitive to the learning rate, so we set learning rate to be smaller, and it perfectly solved the problem.
4. Choice of input: Whether use RGB image or grayscale image could be a problem. When we set the number of layers as 6, grey scale image is 5% more accurate than RGB image. In the meantime, on the side of efficiency, grayscale image converge much faster than RGB image. Therefore we chose grayscale image to be our input.

By doing these experiments, we got our final model shown in Figure 8. The accuracy of the model is more than 92% (human limit 95%, state-of-art 97%).

After the model is well trained, we can use it to convert a segmented sentence image into a vector. The input of the CNN should be a matrix where each row of the matrix represents an character image. Therefore the matrix represents the entire sentence. Each entry of the output vector is the classification result of the each character image.

Given a set of character images in the original sequence of the sentence, the CNN can 'translate' it into a vector. An example is shown in figure 9 below.

[192,2321,432,3054]

Figure 9. An example of character recognition CNN output

Step III: Sentence reconstruction & string segmentation & keyword extraction

Now we have a sentence image expressed as a vector, the next step is to reconstruct the vector back into a sentence string. A reversed dictionary is used in this step. With this dictionary, for example, the vector in figure 9 can be reconstruct to a string '医疗针灸'.

Similar to English letters, a simple Chinese character doesn't have a specific meaning (some of them have meaning and can be used solely, but most of them cannot). Chinese word is a combination of Chinese characters to have an effective meaning. Let's

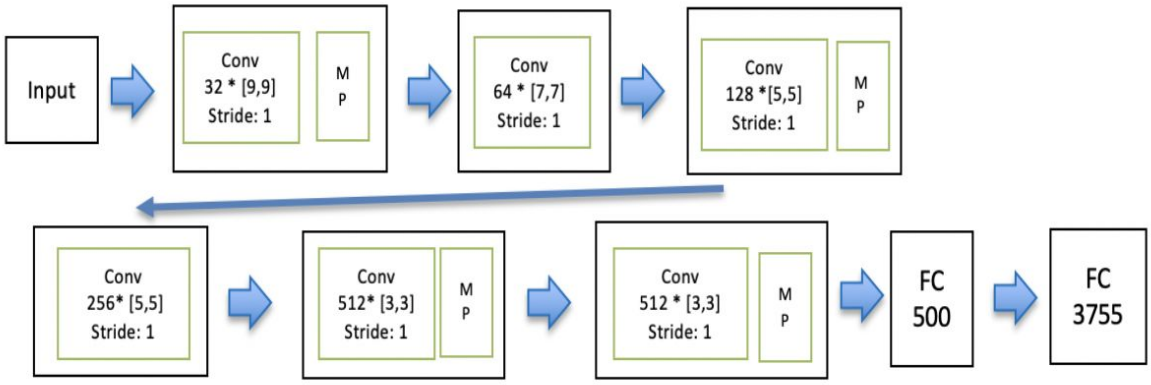

Figure 8. Configuration of character recognition CNN

still use '医疗针灸' as an example. Actually, the example consists of two words, '医疗' (medical treatment) and '针灸' (acupuncture). When classifying an real sentence, we need to segment the sentence into words.

The API we used to do sentence segmentation is Jieba. An example of Jieba' output is shown in figure 10. Algorithms and data structures used in Jieba are as following:

1. A prefix dictionary structure: It has a dictionary consisting of more than 20,000 words and their frequency. It used a trie tree to store these words which can accelerate the speed of searching. To every sentence that needs to be segmented, it build a directed acyclic graph (DAG) for all possible word combinations according to the trie tree.
2. Dynamic programming: Use dynamic programming to find the most probable combination based on the word frequency. For each possible segmentation, it calculated the combination of frequency of every word in it based on the dictionary. And finally find the most probable combination.
3. HMM-based model with the Viterbi algorithm[*1]: To deal with the words that are not registered in dictionary.

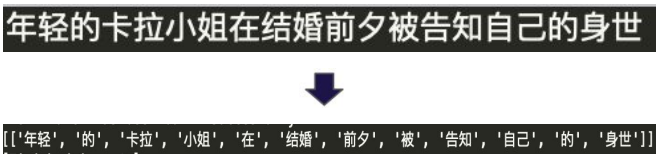

Figure 10. An example of Jieba output. Splits a string of sentence into a list of words

Given by the developer, Jieba has more than 85% accuracy in sentence segmentation.

A word-to-int dictionary (Similar to word2doc) is provided by Sougou, which supports more than 1 million word records. We used it to convert the output list from Jieba to a vector of integers. For words which has no record in dictionary, it will be marked as 0. Therefore, the variance between difference can be very small if a lot of words are not in record. We will give a more detailed analysis on this in the section 5.

In real sentences, there are a lot of common and repeating words, just like 'he', 'she', 'and', 'but' in English. These words are bad for sentence classification because they will weaken the difference between sentences. Considering this fact, our philosophy is to classify the sentence by its key words. Therefore, we need to extract keywords from each sentence.

Term frequency–inverse document frequency (tf-idf) is widely used in keyword extraction. A key word of a specific class of sentence should have a higher frequency in this class but lower frequency in the entire sentence dataset. The formula of tf-idf is given below:

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

Where tf(t,d) is term frequency and idf(t,D) is inverse document frequency.

Step IV: Semantic Classification

A pre-trained convolutional neural network designed by Sougou which support 56 semantic classes for news classification is used here. Python libraries involved are tensorflow, keras, and numpy.

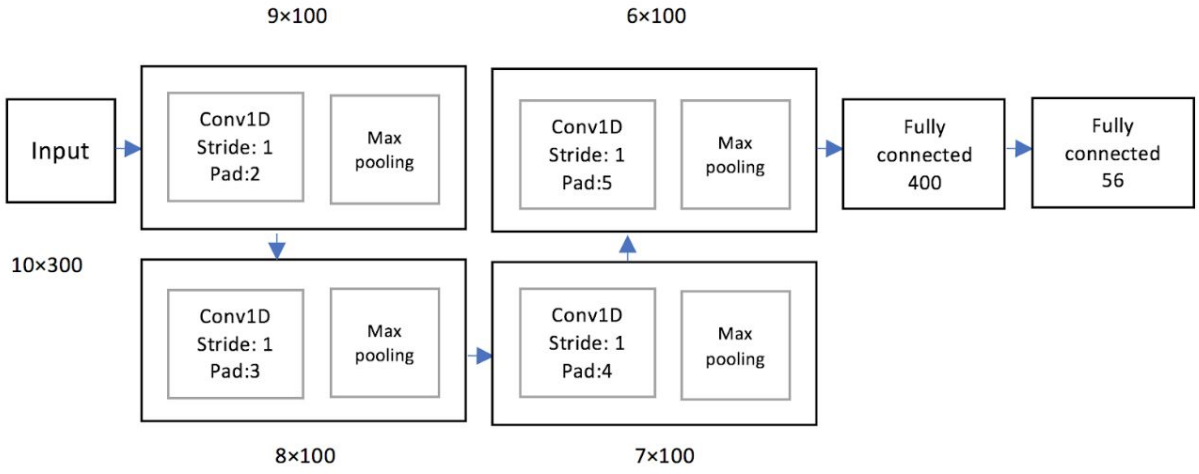

Figure 11. Configuration of semantic classification CNN

The network consists of 4 convolutional layers, each of them has 100 filters. Each convolutional layer is followed by a pooling layer and a flatten layer. The output then enters a fully connected layer to compute the possibility of being a class in 56 semantic classes. The matrix we used here for loss is cross entropy, for accuracy is corrected classified rate (CCR), and the optimizer is Adam. Figure 11 shows the CNN configuration.

Considering our goal is to classify a sentence, but not an article, before training the model, we firstly shortened each training sample by randomly picking up part of the keywords from the samples in the same class. We tried to set the keyword number as 10, 50, 100, 200 and 400 got our best performance on **training set** at 400 keywords. Detailed performance is in section 5.

As a matter of fact, there's no appropriate handwritten Chinese sentence dataset online, so we generate the dataset by ourselves. We will talk about it in section 4.

Connecting each step together, the procedure of our final classification is:

Given a image of handwritten Chinese sentence → Segment the image of sentence into multiple images of characters → Classify the images into 3755 classes, output a vector → Translate the vector to a Chinese string → Segment the string to a list of words, and convert to another vector → Classify the vector into 56 semantic classes

4. Dataset

I: Sentence segmentation

As mentioned in Section 3 Step IV, we generated our dataset. By randomly picking up images from CASIS-HWDB1.1 (see details in part II: Character recognition), and concatenate them horizontally, we created 10000 samples of sentence from with difference in length. There are 3755 characters in test dataset totally. We generated test sentences in length 6, 8, 10, 12 and 14 separately. Each case includes 2000 images.

II: Character recognition

The dataset we used for character recognition was CASIS-HWDB1.1, it is developed by the National Laboratory of Pattern Recognition, Institute of Automation of Chinese Academy of Science. Details of the dataset is shown in table 1.

Dataset	Writers	Classes	Samples	Valid Samples	Training 80%	Test 20%
CASIS-HWDB1.1	300	3755	1,121,749	1,121,735	897,748	223,987

Table 1. CASIS-HWDB1.1 dataset information

Figure 12 is a sample of one class of characters in the dataset.

Figure 12. The same character written by different writers in CASIS-HWDB1.1

However, the original dataset is in gnt file, and it is not feasible to our CNN input format. What's more, the input image size should also be unified. We did 2 steps of preprocessing on the dataset:

1. We converted the gnt file to png file.
2. We adjusted the size of input images. The images in the dataset are in different shapes, so we added white spaces to either top and bottom or left and top to make them in square size, and resized to make them in the desired size (58×58).

Python library involved is OpenCV2.

III. Semantic classification

The dataset we used here was a compilation of 179487 news from Chinese news portals, such as Today's headlines, Sina, Baidu News, etc. Collected by Sougou.

Dataset	Classes	Samples	Training 80%	Test 20%
Sougou News Compilation	56	179,487	143,590	35,897

Table 2. Sougou News Compilation dataset information

As mentioned in section 3 Step III, the dataset was preprocessed with Jieba and tf-idf:

1. Segment strings into lists with Jieba.
2. Extract keywords and rebuild the list with tf-idf.

IV: Final test

We randomly picked up 10 keywords from preprocessed Sougou dataset, and encoded each character in those keywords by a character-to-int dictionary (such as '{'你':1, '我':2}'). Then we found out the corresponding handwritten character images from

our preprocessed CASIS-HWDB1.1 dataset (every character image is grayscale 58×58), as the dictionary is built consistently with the class numbers. We concatenate the images horizontally, and then generate a data like figure 13.

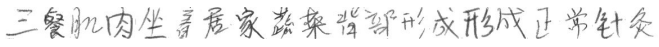

Figure 13. An example of test sentence (To be noticed that this is not a real sentence, it's just a combination of keywords)

With a consideration of our computer storage and running time, we generated 20 'sentences' for each class (56 classes totally), so the dataset has 1120 'sentences' totally. Table 3 concludes the information of the dataset.

Dataset	Classes	Samples	Training 80%	Test 20%
Self-generated	56	1120 (56*20)	896	224

Table 3. Self-generated dataset information

5. Results

I: Image segmentation

We tested our program on 10000 samples. The segmentation results is shown in table 4. In table 4, result means the difference of the number of characters between the output and the truth. For example, if we only get 9 segments from a sentence with 10 characters, it is a "-1" result. Even though, we can't regard "0" as a absolute correct result, because it can get correct number of the characters, while segments it in a wrong way. To check if every segment is correct, we can put it in a CNN for further validation.

result	0	-1	+1	+2
amount	9784	31	184	1

Table 4. Performance of sentence segmentation

II: Character recognition

The metric we used for character recognition was correct classification rate, defined as the number of samples that are correctly classified divided by the total number of test samples.

As shown in Table 5, when the network is has 6 convolutional layers, the model outputs the best test performance 92.03%.

Conv Layers	4	5	6	7
Accuracy	84.64%	90.07%	**92.03%**	91.67%

Table 5. Test accuracy against the number of convolutional layers

To better analyze the result, we plot the training and test accuracy and loss as shown in Figure 14, we can see the loss and accuracy converges fairly good with the growth of epoch number.

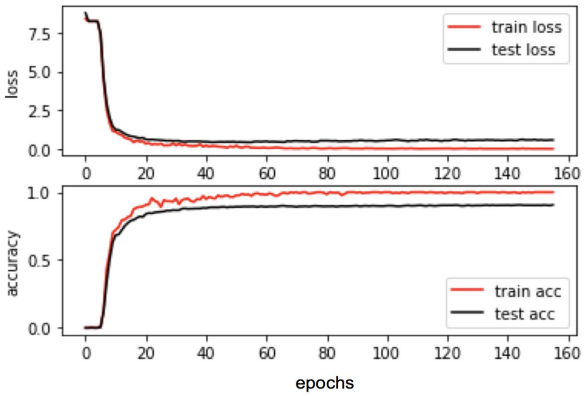

Figure 14. Training, test accuracy and loss against epoch

The training accuracy is a little higher than test accuracy and the training loss is still decreasing while test accuracy converges, which implies that the model is slightly overfitting.

III: Semantic classification

The metric we used for character recognition was still correct classification rate.

As shown in Table 6, after the keyword number exceeds 50, the accuracy almost keeps above 99%, and the best accuray is 99.44% when we set the keyword number as 400. This is easy to understand because a higher model complexity will fits the training dataset better.

Number of Keywords	10	50	100	200	400
Accuracy	87.07%	99.00%	98.88%	99.10%	**99.44%**

Table 6. Training accuracy against the number of keywords

However, we noticed that the distribution of training dataset is not even for each class. A histogram of statistics of the number of samples in each class is given in figure 15 below. We can see that most of the classes have less than 1000 samples in them, and we found number of sample in each class varies from 30 to 30000.

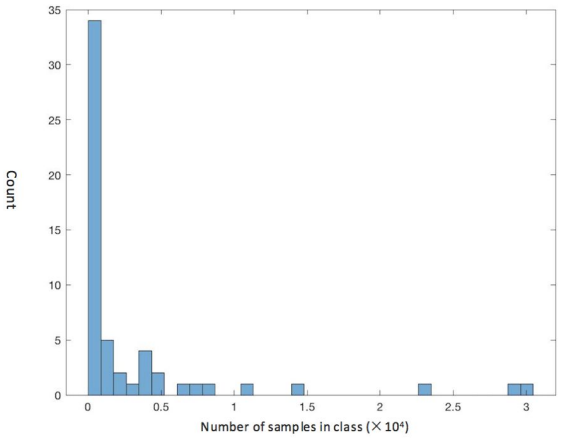

Figure 15. Distribution of sample amount

We also checked the classification accuracy for each class, and we found the accuracy of small classes is obviously lower than large classes. Some of the small classes even have 0 accuracy. However, this can't be observed from the entire performance, as large class have absolutely more contribution to the entire performance, and blankets the poor performance of small classes.

IV. Final test

The metric we used for character recognition was still correct classification rate. Table 7 shows our model's performance with considering different number of keywords.

Number of Keywords	10	50	100	200	400
Accuracy	45.89 %	56.88%	58.13%	**65.45%**	63.66%

Table 7. Test accuracy against the number of keywords

The final performance of The final result is not as good as we have expected. The best accuracy of the classification on 1120 test 'sentences' is about 65% when we set the keyword number as 200.

Our analysis about the performance includes:

1. Accumulated error: As there are some steps in our model, and every step has a possibility to make a mistake. The mistake may be even augmented during the steps.
2. Keyword number choice: Performance of choosing a small number of keywords' in each sample when training is very limited because it may not cover keywords appear in test samples. As we can see in table 2, 10 keywords' accuracy is the lowest one. A large number of keywords in training is also not a wise choice because overfitting grows with the model complexity. What's more, as our test samples only have 10 keywords, to have the test sample size fit the model, we have to repeat the test sample (for example, we need build a 1×400 vector by repeating the 1×10 vector for 40 times). This operation will reduce the variance between different test samples.
3. Training sample distribution: As mentioned in Section 5 III, the distribution of training set is not even. However, our test set is generated evenly (20 samples per class). Therefore, in our model, the poor performance of small classes is no more blanked by large classes. This is the most important reason of the great reduction in performance.

6. Detailed Roles

Task	File names	Who
Image segmentation	File Sentence_Segmentation	Ganquan Wen
Wrote first version for image segmentation; Segmented sentences for semantic classifier model	File Version 1; File text_seg	Jiali Ge
Wrote test code for image segmentation; Generated test sentences	Segmentation_test.py; Sentence_generator	Chenyu Wang
Built character classifier model	File character classifier	Yuchen Wang
Built semantic classifier model	File text_cnn/semantic classifier	Runzhou Han
Wrote section 1,6 for report		Jiali Ge
Wrote section 2 for report		Chenyu Wang
Worte section 3 for report		All
Worte section 4-5 for report		Runzhou Han, Yuchen Wang, Ganquan Wen

7. Code repository

https://github.com/GanquanWen/BU-EC500-Deep-Learing-Project

All of our code is in the repo above. Our programs for sentence segmentation, character recognition, dataset generator and data preprocessing are all original. For semantic classification, it is adapted from Sougou's code (one of us participated in Sougou summer intern 2018, and this is part of his work. Unfortunately, there is no available link to access the entire work so far). Follow the instruction in Readme file of our repo, you can reproduce our model. Have fun!

References

[1] Yu N, Jiao P, Zheng Y: Handwritten digits recognition based on improved LeNet5[C]//control and decision conference on. IEEE, 2011: 1135-1139

[2] Sajjad S. Ahranjany ; Farbod Razzazi ; Mohammad H. Ghassemian: A very high accuracy handwritten character recognition system for Farsi/Arabic digits using Convolutional Neural Networks. 10.1109/BICTA.2010.5645265

[3] Dan Claudiu Ciresan ; Ueli Meier ; Luca Maria Gambardella ; Jurgen Schmidhuber: Convolutional Neural Network Committees for Handwritten Character Classification. 10.1109/ICDAR.2011.229

[4] Zhang Y. *Deep Convolutional Network for Handwritten Chinese Character Recognition*. 2015.

Appendix

*1. HMM-based model with the Viterbi algorithm: it uses HMM-based model with the Viterbi algorithm. In HMM model, it uses four labels to mark Chinese word, B-- the word is in the position of 'begin'; E-- the word is in the 'end' position; M-- the word is in the middle; S-- the word is single. According to the probability matrix and Viterbi algorithm. We can get a most probable 'BEMS' sequence for a sentence. According to the rule that B is the beginning and E is the ending of a word, a final segmentation can be achieved.