Handwritten Kanji Recognition

Deep learning project work – Márton Zalavári

Main goal

The task is to recognize handwritten Kanji and interpret them as digital UTF-8 characters.

The Data

I use the ETL Character Database which contains numerous handwritten characters, so it is good for starting. It is free to download for research purposes. [1]

For the time being I only used the ETL8G dataset, which contains the 881 most commonly used Kanji (and 75 additional Hiragana) written by 160 different people. Each image consists of 128(X-axis size) * 127(Y-axis size) pixels, and each pixel is encoded on 4bit (16 gray level) [2]

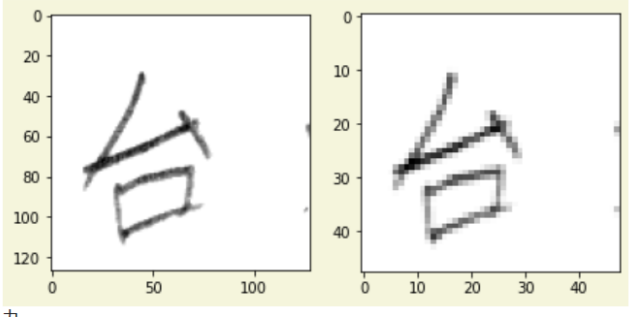
I want to highlight, that there are some other solutions (linked at the references), which is similar to my work, but even if some code is similar, I did more research, changed, fixed and simplified plenty of things. [3]

Reading and pre-processing the data.

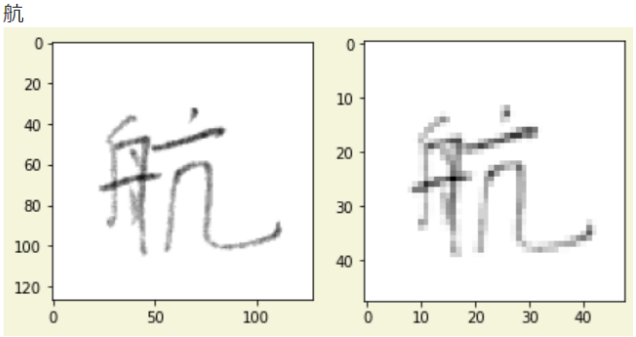
I read the datasets and load them into python code. I filter out the 75 hiragana character and use only the Kanjis. After reshaping the arrays, I make appropriate labels for each image. Then I needed some image processing. The images originally were 128 pixels in width and height, and each pixel had 4bit greyscale information. I downscaled the images to have 48 times 48 pixels in size. This was like a compromise, with this the network did not need to be too big, the memory was sufficient, and the characters were still recognizable. I tried 64 pixels, and 32 pixels, both did work, but the results were not so good.

Some example is shown below. We have the actual Kanji, a corresponding image extracted from the database, and the same image downscaled. It can be observed that minor details can be blurred in the downscaled image, but the character is still fairly recognizable.

台



航



Creating and training a neural network

Creating the labels for the data, splitting the dataset was straightforward.

Then I used data-augmentation on the images by applying slight rotations, zooms and shearing.

The network itself is simple, I use 3 convolutional layers, each of them followed by a maxpooling layer. Then I add a dense layer and another dense layer as the output layer. In the end, I had about 1 million trainable parameters.

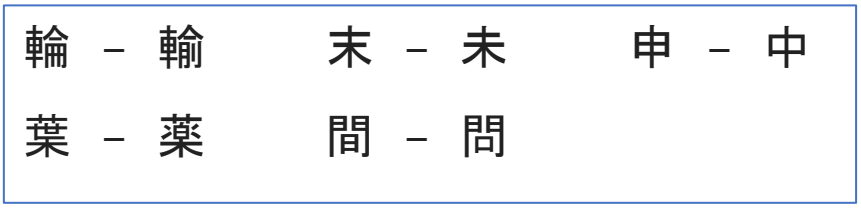
I did some experiments with different hyperparameters, these settings worked the best. I mean sometimes I got better results, but the model was either much larger or the training process slower etc. I came across a study, where they tried out different network structures for (Chinese) character recognition, and their inspection was also that more layers barely achieve better accuracy. It is better, but the difference is minimal. [4]

Results

The training process is relatively slow (takes about half an hour), but even after 5 minutes we are able to reach 95% accuracy. At the end of the training process usually more than 99%.

I plotted the confusion matrix, but that is not very informative, because we have too many classes. I also calculated the precision and recall for each character. Most of them are near-perfect.

While evaluating the results, I checked the most misclassified elements, to find out that the predicted characters are looking really similar to the ground truth, sometimes they are differing only by one stroke.



Applying the model

I created a proof-of-concept application, where you can draw a character on a canvas, and it will be guessed by the computer in real-time, which was the character drawn. I used some HTML and JavaScript, but it is currently working only from the Colab environment.

References

[1] Electrotechnical Laboratory, Japanese Technical Committee for Optical Character Recognition, ETL Character Database, 1973-1984.

[2] http://etlcdb.db.aist.go.jp/etlcdb/etln/form_e8g.htm

[3] https://github.com/Nippon2019/Handwritten-Japanese-Recognition

[4] Yuhao Zhang „Deep Convolutional Network for Handwritten Chinese Character Recognition”