A small system for recognizing Chinese characters

Dataset

I used the AI FREE Team's handwritten traditional Chinese characters dataset, which contains - on the whole - '13,065 different Chinese characters, with average of 50 samples for each character'. The characters are stored in `.png` format and have the width and height of 300. To speed up the task - while not drastically decreasing its complexity - I removed a part of the dataset - to be precise, all the characters whose numbers began with digit 1, comprising about 30% of the dataset. A dataset of this size (but a more reasonable selection of characters) would be enough for most practical purposes, as an educated Chinese person typically knows about 4000 characters.

Model architecture

The image, after preprocessing - described below - is basically processed by a convolution module, a multi-head self-attention module and a fully connected affine layer, followed by a final affine transformation to logits. The more precise data flow is: twodimensional convolution, ReLU nonlinearity, twodimensional max pooling, flattening each channel's 2D representation, multihead attention between channels, flattening again (concatenating all channels; representation after attention), an affine transformation, ReLU lonlinearity, and the final transformation to logits. I decided to use the convolution module as a common method of capturing two-dimensional relationships, and then attention because I hoped it would facilitate capturing long-distance relationships, like a stroke spanning most of the image.

Such a model has many adjustable hyperparameters. Unfortunately I did not have the time to play around with those too much; I settled for ones that are typical or seemed reasonable. Thus batch size is 32, learning rate $1e-3$, images are scaled to height and width of 64 during preprocessing, there are 100 epochs, 15 attention heads, 16 convolution channels, kernel size 3, stride 2, and max-pool kernel size 2. Stochastic gradient descent was used as the optimizer.

Preprocessing

All images were scaled to height and width of 64. In addition, they were binarized, with grey pixels converted to value 1, and white ones to 0.

Results

The model obtained an accuracy of 29195 out of 46671 - about 63%. It should probably be noted that this is validation accuracy, rather than test accuracy. The dataset was split into training and validation splits, and in each epoch, it was trained on the train split, and evaluated on the validation split. Then the model with the highest validation accuracy was chosen out of the checkpoints for each epoch. This way, the result might not be fully free from overfitting, and results on a separate test set might be slightly worse.

Second model variation

After training the initial model, I added some preprocessing. Before resizing, images bigger than 64×64 were trimmed, i.e. the rectangular area actually occupied by the character was selected, and the rest of the image, containing only white pixels, was discarded. This serves to unify the representations of characters differing only in size, or on their location in the image. If it is still necessary to resize down, the impact of low resulution is also alleviated, since the images are resized from a smaller size. Afterwards, thinning was applied to achieve a more uniform stroke width (in Chinese characters the shape of the strokes matters, but their width does not).

After applying these changes, I noticed a problem - the rate of validation loss decrease slowed down dramatically. To be honest, I am not sure what caused this issue. Increasing the learning rate to $1e-2$ did not seem to work. However, I kept this change and finally resolved the problem by reducing model complexity, i.e. removing the attention layer

altogether. The accuracy increased to 33172 out of 46671, i.e. 71%. Unfortunately I did not have the time to conduct and ablation study and check what the actual influence of the preprocessing was, i.e. what the accuracy would be for a model without the attention unit, but also without preprocessing.

Suggestions for improvement

In recent years, it has become common in Natural Language processing to enrich deep learning models with domain-specific knowledge. An example of this are dependency-tree-based convolutional neural networks [1], utilizing semantic information based on dependency trees. For this task, it might be possible to increase accuracy while reducing computational cost by decomposing it into two subtasks, utilizing the characteristics of Chinese characters. The crucial characteristic is that they are composed of smaller subcomponents, which are either so-called *radicals*, or can be recursively segmented further. Furthermore, there are a finite number of ways in which these smaller components are composed together to create a bigger character (such as stacking top-down, left-right, surrounding etc.). Finally, the number of radicals is much smaller than that of characters, mostly given as 214. Therefore, following the lightweight model in [2], we could first segment a character into its subcomponents, and then classify each of those independently using a small model (with only 214 class, rather than thousands as in the model recognizing whole characters). Finally we would compare the results against a database storing - for each character - its decomposition into radicals, and try to find an exact match or at least the most similar character.

References

[1] Yuhao Zhang, Peng Qi, Christopher D. Manning. Graph Convolution over Pruned Dependency Trees Improves Relation Extraction. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018

[2] Yongtang Bao, Yue Qi, Bowen Yu. A Component Recognition-based Chinese Character Segmentation and Structure Discrimination Method. Proceedings of the 2015 International Conference on Computer Science and Intelligent Communication, 2015