OCR for Handwritten Chinese Characters using CNN

By Cary Wu and Hudson Chou

Abstract

The paper evaluates an optical character recognition (OCR) system that aims to recognize handwritten Chinese characters and output a machine-encoded text character. OCR is the process by which images, pdfs, and scanned documents are converted into data that is understood by the computer and can be easy to work with. The motivation for this project is to create on OCR system for Chinese so that humans can easily input unknown handwritten text characters into softwares such as Pleco, Google Translate, and search bars for translations and search results. A 3-layer convolutional neural network (CNN) is the model that was used to train the data to classify images of handwritten characters to their respective labels. We chose to use 70 labels/characters to have the system recognize and classify. After training the model, we ended up with an accuracy of .93 on the testing set.

Introduction

When it comes to handwritten Chinese, various handwriting styles can sometimes be difficult for human eye to recognize and confusion between similar characters is not uncommon. With a well trained OCR system for Chinese, a simple scan or picture of some Chinese characters can conveniently be used to input into translation softwares for people who struggle to understand written Chinese. We train handwritten samples of chinese characters using a convolutional neural network and then use the model to predict handwritten characters and output the associated label in text form.

Dataset

Our dataset is a subset of the larger dataset provided by the National Laboratory of Pattern Recognition (NLPR) and Institute of Automation of Chinese Academy of Sciences (CASIA). We focused on the offline isolated characters dataset which are in the files named HWDB1.1trn_gnt_P1, HWDB1.1trn_gnt_P2, and HWDB1.1tst_gnt [2]. For our datasets provided by NLPR and CASIA, the original data samples already went through feature extraction. Unfortunately, we lacked the computing power to work with all of the samples and classes of characters available due to lack of time and computing processing power. Whereas the original database consisted of 3,755 classes of characters and a total of 300 different writers for each class, we only used 70 classes and 228 (train 118 + validation 55 + test 55) different writing samples per class. *The **70** classes of characters we chose are listed:*
['祝', '铰', '联', '瑟', '坑', '挑', '研', '件', '哀', '静', '儒', '藐', '瞪', '拿', '赡', '逞', '疫', '蜘', '山', '舟', '训', '圣', '毁', '舶', '煌', '骂', '假', '轮', '谍', '榆', '鼎', '硅', '眠', '朴', '半', '迄', '促', '绎', '觅', '勺', '戳', '酵', '撕', '盂', '弘', '忘', '砧', '蝴', '燎', '题', '尧', '谭', '订', '矗', '襟', '踪', '麓', '规', '挨', '恤', '聪', '卞', '丁', '岔', '租', '充', '各', '猴', '缩', '枚']
Furthermore, we utilized an open-source python library called PyCasia and some additional preprocessing methods by Lucas Kjaero [3][4] to properly download and load the datasets. We extracted 118 samples of each of the 70 characters from HWDB1.1trn_gnt_P1, and 55 samples

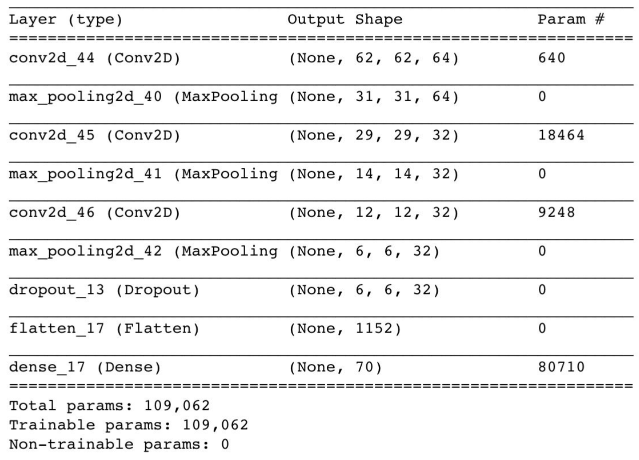
samples of each character from HWDB1.1trn_gnt_P2 and HWDB1.1tst_gnt to use for validation and testing, respectively. A summary of the subset dataset we used to train our model is in Table 1**.** Detailed explanation on the dataset can be found in the paper describing the database [1]. Furthermore, we resized each grayscale image into a 64 x 64 dimension image and represented it as a normalized matrix.

Table 1. Description of datasets

File Name	Purpose	Number of samples per label	Number of features (after resizing)
HWDB1.1trn_gnt_P1	Training	118	64 x 64
HWDB1.1trn_gnt_P2	Validation	55	64 x 64
HWDB1.1tst_gnt	Testing	55	64 x 64

Model

Our model for CNN consists of 3 layers with Max Pooling, and Dropouts. Max pooling is inserted after every layer to decrease the size therefore making our model run faster and more efficiently. In addition, we added the Dropout module to ensure overfitting would not occur. In order to make certain that our model would work in recognizing images of various shapes with different amounts of spacing, we implemented data augmentation using keras' image data generator. The image data generator manipulates the input data by resizing the original images to ensure the training data is varied.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_44 (Conv2D)           (None, 62, 62, 64)        640
_________________________________________________________________
max_pooling2d_40 (MaxPooling (None, 31, 31, 64)        0
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 29, 29, 32)        18464
_________________________________________________________________
max_pooling2d_41 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 12, 12, 32)        9248
_________________________________________________________________
max_pooling2d_42 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
dropout_13 (Dropout)         (None, 6, 6, 32)          0
_________________________________________________________________
flatten_17 (Flatten)         (None, 1152)              0
_________________________________________________________________
dense_17 (Dense)             (None, 70)                80710
=================================================================
Total params: 109,062
Trainable params: 109,062
Non-trainable params: 0
```

After testing both ADAM and RMSProp, we concluded that RMSProp had a much faster runtime with a much higher accuracy. While ADAM and RMSProp both utilize Stochastic Gradient Descent, unlike ADAM, RMSProp balances the step size by dynamically changing gradient.

The loss function we used in our model was Categorical Cross Entropy. Cross Entropy is used to describe the difference between two probability distributions. Given a one-hot-distribution, CCE calculates how much the model's predicted result varies from the actual intended result.

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{1}_{y_i \in C_c}\log p_{model}[y_i \in C_c]$$

Formula for Categorical Cross Entropy

$$Acc = \frac{\text{\# of correct predictions}}{\text{total number of samples}}$$

Formula for determining our prediction accuracy

Results

Our model had an accuracy of .93 and a loss of .25 after training it over 8 epochs using the model described above. It is able to accurately predict most characters and works on characters outside the dataset as well as long as the character fills the image as the main input. However, our model struggles with images that contain too much space in the margins. This issue can potentially be solved by implementing OpenCV.

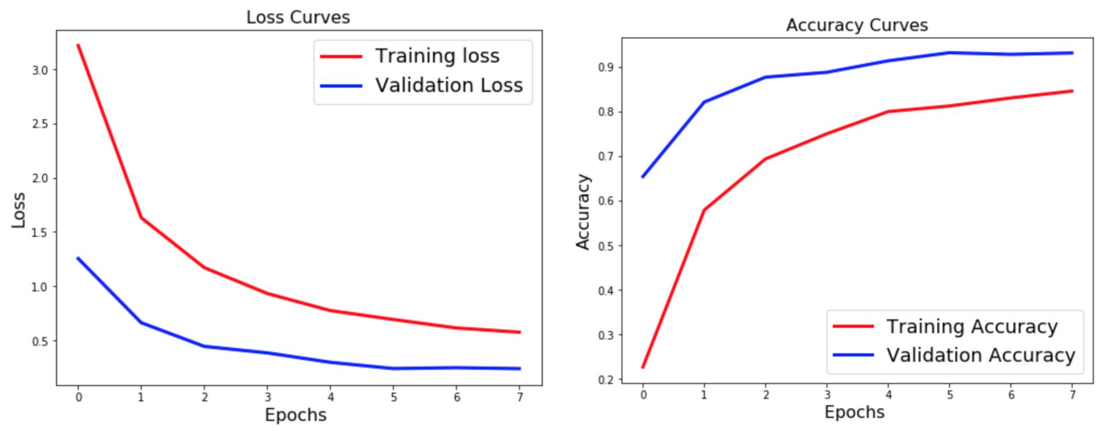


Figure 1. Accuracy and loss curves for training and validation datasets.

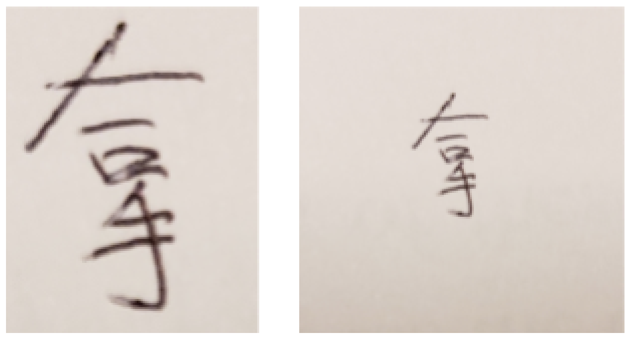


Figure 2. The model correctly identifies the image on the left but incorrectly identifies the image on the right due to the increased margin space of the second character

Future Work/Conclusion

Discrepancies in handwriting styles and the presence of characters that look extremely similar remain as challenges still facing handwritten chinese character recognition as shown in Figure 3.

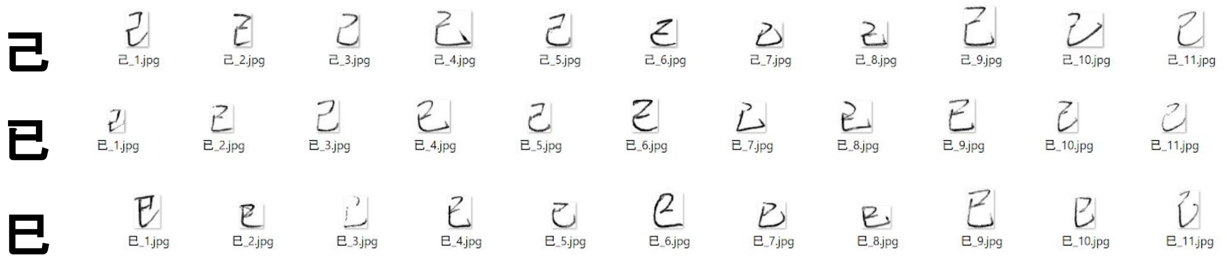


Figure 3. Similarities in the appearance of chinese characters and different handwriting styles for three very similar looking characters (己 Jǐ , 已 Yǐ, 巳 Sì). Note that "_1.jpg" means that it was written by the same writer.

Furthermore, beyond simply recognizing individual characters, it is much more useful and rewarding to recognize sentences, paragraphs, and full on texts written in Chinese. With that, fine tuning the preprocessing and feature extraction techniques will pay a crucial role. Integrating OpenCV with our project is a future goal that would allow us to extract characters from natural images in addition to more accurately identifying images with large margins around the characters. Of course, with a stronger computer that uses a GPU, training time would've been greatly reduced and thus, we would've been able to train more samples, play around with more layers in our CNN, run more epochs, etc.

References:

[1]Liu, Cbeng Lin, et al. "CASIA Online and Offline Chinese Handwriting Databases." *National Laboratory of Pattern Recognition (NLPR)*, www.nlpr.ia.ac.cn/databases/download/ICDAR2011-CASIA databases.pdf.

[2]"CASIA Online and Offline Chinese Handwriting Databases." *Offline Database*, www.nlpr.ia.ac.cn/databases/handwriting/Download.html.

[3]L. Kjaero, PyCasia, (2017), GitHub, https://github.com/lucaskjaero/PyCasia

[4]L. Kjaero, chinese-character-recognizer, (2017), GitHub repository,
[5]Gupta, Vikas. “Home.” *Learn OpenCV*, 29 Nov. 2017,
www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/.
[6]Bouda. “Day 69: Rmsprop – 100 Days of Algorithms .” *Medium.com*, Medium, 1 June 2017,
medium.com/100-days-of-algorithms/day-69-rmsprop-7a88d475003b.
[7] Sebastian Ruder. “An Overview of Gradient Descent Optimization Algorithms.” Sebastian Ruder, Sebastian Ruder, 29 Nov. 2018,
ruder.io/optimizing-gradient-descent/index.html#rmsprop.