CS230

Convolutional Neural Network for Handwritten Chinese Character Recognition

Ze Wang*
SCPD
Stanford University
ze0330@stanford.edu

Abstract

This project is focused on using neural network to recognize handwritten Chinese characters, where a five layer baseline modal was built from scratch based on the code-lab on coursera and a Convolutional Neural Network(CNN) was built based on an open-source GitHub projects. Different optimizations of bias and variance were conducted on the CNN models. The optimal 14 layers CNN model reached an accuracy of 95.3% on HWDB1.1tst test set from CASIA[1]

1 Introduction

Compared to recognizing handwritten digits and English alphabets, Chinese character recognition is a more challenging task. There are more than 3000 frequently used Chinese characters while there are only 10 digits and 26 English characters. Also, Chinese characters are Hieroglyphs whose structure are more complicated and included way more details.

The input of out model is .gnt format file downloaded from CASIA offline database. (Specifically, HWDB1.1trn and HWDB1.1tst) http://www.nlpr.ia.ac.cn/databases/handwriting/Download.html. The output of our model is an integer number ranging from 0 to 3754. Each output number will represent a character in the dictionary. In this project, the final model is a fourteen layers CNN model. The model was trained on a AWS EC2 GPU instance. The implementation was based on python and tensorflow. After fine-tuning the model, it achieved an accuracy of 95.3% on the HWDB1.1tst test set from CASIA. And it's a solo project. I chose to stay on character level based on my background and maximum time I can contribute to this project. And it gave me a really good hands on experience to train a model myself.

2 Related work

This project baseline model was from the paper by Xuefeng Xiao et. al. [2] – Building Fast and Compact Convolutional Neural Networks for Offline Handwritten Chinese Character Recognition In this paper, they designed a nine layer of CNN for 3755-character classes. Although this paper is focus on the performance (memory and CPU utilization), their baseline model (without any performance tunning) reached an accuracy of 97.30 1.0, which can represent the state-of-art level of this problem. I took the nine-layer model structure out, and reproduced this model structure in milestone 2. It's not a complicated model compared to other literatures, but it's concise enough to start with CNN for a

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

CS230: Deep Learning, Spring 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

beginner like me. And it has very detailed matrixes in the baseline model experiments to compare with. That's the main reason for me to choose this paper.

Other works such as the CNN in paper [3], it's also a CNN based model, tested on the same data set. This CNN model has less conv layer and one extra fully connected layer. It reaches an accuracy of 95.5%, 0.2% higher than what I present in this paper. However, the implementation details are not very specific in this paper. Another interesting paper [4], it used pre-trained GoogLeNet and directional feature maps for feature extraction. And it reached accuracy over 96%. I was initially want to implement this version of CNN, however, it's hard to find open source support for this and there is not enough time. But the pre-trained GoogLeNet should definitely be tried.

3 Dataset and Features

The data source comes from the CASIA offline database. (Specifically, HWDB1.1trn and HWDB1.1tst) http://www.nlpr.ia.ac.cn/databases/handwriting/Download.html.

The data downloaded part was referenced to open source code in [5]. The data processing part, from the raw .gnt format to .png image. One changes made during the decoding process is that, the decoding schema was gb2312, but it failed to decode some of the original data in the test set. Therefore, the decoding schema got replaced with gb18030, which is also backward compatible with gb2312. Full dataset of HWDB1.1trn and HWDB1.1tst were generated and uploaded to EC2 instance. In total, there are 3755 classes in total. 22391 .png files in the testing set, avg. 59 per class, 897445 .png files in the training set, avg 239 per classed. Examples are show in fig 1.

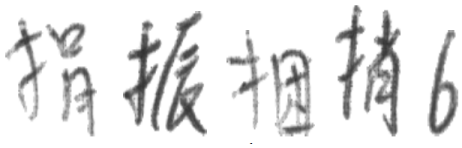

Figure 1: Input data examples

4 Methods

The original nine-layer structure was a few combinations of convolutional layer and max pooling layers, followed by a fully connected layer and then output layer. This nine-layer schedule is normal among literatures in this topic. In real implementation, it got extended to fourteen-layer structure model, seven convolutional layers and five max pooling layers followed by a fully connected layer and an output layer. CNN model layout as fig 2.

Batch normalization and l2 regularization were used in each CONV layer to accelerate the computation process. And in the last fully connect layer, a dropout probability of 0.5 was used in the baseline model. And in the baseline model, a basic momentum optimizer was used but still achieve a great accuracy with learning rate decay. The CONV layer channel number and the fully connected layer length are all pre-defined, should be used in later experiments to see the loss. The implementation was based on tensorflow_p36 environment on p2.xlarge instance. The baseline model took about one hour and thirty minutes to train, and less than a minute to test.

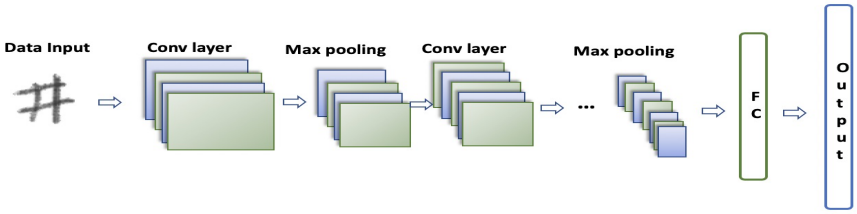

Figure 2: Model layout

5 Experiments/Results/Discussion

The training loss, top 1 position accuracy and test accuracy are three major parameters I used. Other hyper-parameters are compared during the training process.

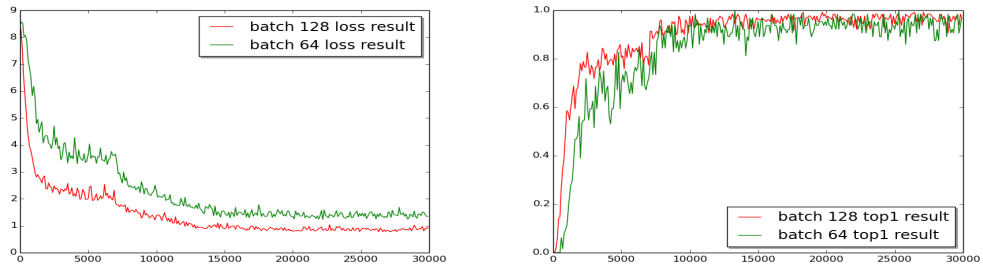

Figure 3: training loss and top 1 accuracy for batch size diff

From fig 3, we can see that the batch size of 128 did much better than batch size 64.

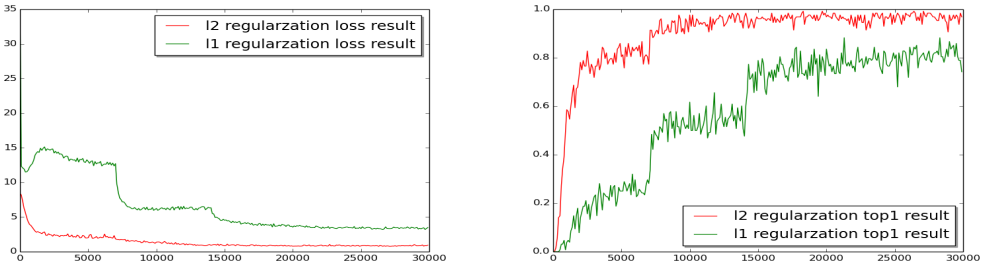

Figure 4: training loss and top 1 accuracy for l1 and l2

L1 regularization tends to focus on a relatively small portion of features compared to l2 regularization. And it seems obvious that, this task of recognizing the handwritten Chinese character is more sophisticated, and need more features to capture the details. Thus L2 regularization performs better in this case.

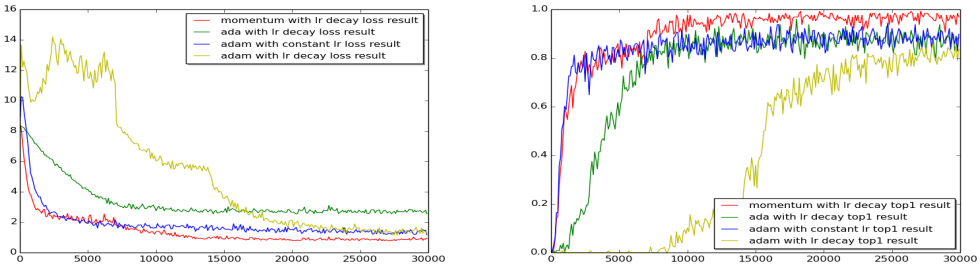

Figure 5: training loss and top 1 accuracy for optimizer diff

The one of the most important experiment I made was trying to see the diff between different optimizers. Here I chose four optimizer – momentum, AdaGrad, Adam with constant learning rate, and Adam with learning rate decay. All of those four experiments has a base learning rate of 0.1. For learning rate decay, the decay factor is always the same as the base learning rate. Everything else is the same. Here comes the result are shown in Figure 5. It seems that momentum has the best accuracy. And from the test result, momentum also has the best accuracy of 95.1%. The adam optimizer should be the best optimizer among those, therefore, it means the optimal learning rate for adam optimizer are not found. Instead, I did an experiment with the adam optimizer with different rate.

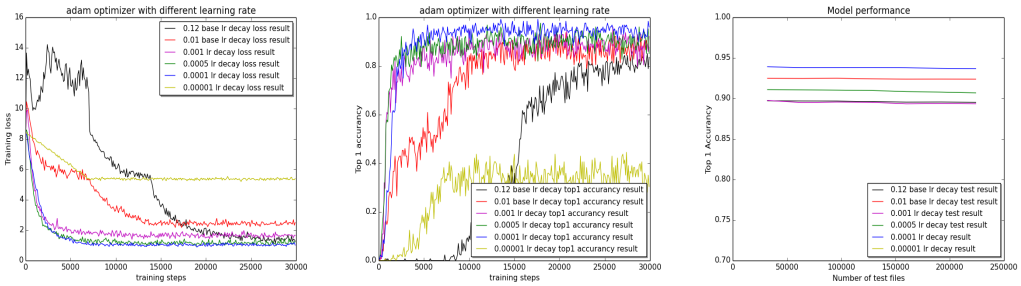

Figure 6: training loss, top 1 accuracy and test result for adam

After tunning the learning rate on exponential scale, the base learning rate of 0.00001 was found to be the best learning rate for adam optimizer.

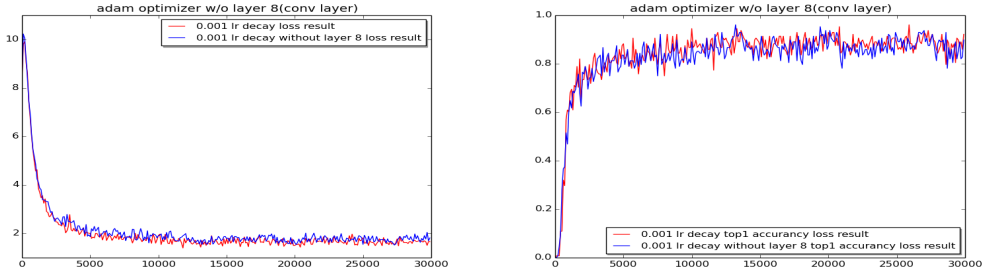

Figure 7: training loss, top 1 accuracy w/0 CONV layer 8

Apart from hyper parameters tuning, I also tune the structure of this model by taking one CONV layer (layer 8) out, and compared the effect with everything else the same. The difference is subtle. From Figure 7, we can see that, with one less layer, model performs less well. It's expected, because it means we are gonna loss some details brought by layer 8. And it echos the result we see from the L1 and L2 regularization, more features and details are need to bring up the model accuracy.

6 Conclusion

In this project, the baseline neural network mode I built in milestone 1 reached a test accuracy of 73.01% for the test set. The CNN model, after fine tuning, reached an accuracy of 95.3 %. The best accuracy comes with the original 14 layer CNN structure, momentum optimizer, 0.5 dropout, L2 regularization, and 0.1 based learning rate decay. Although the adam optimizer is superior than other optimizer, the optimal learning rate for adam optimizer is still not found. And I have the suspension that adam doesn't work well with learning rate decay in this model.

7 Future Work

More works could have been done in the future regrading this. Moreover, after the successful result of character level recognition, a handwritten text level recognition using sequence model should be explored in the future as well.

The main purpose of this project is to get me finish the first deep learning project all by myself. And the use of AWS GPU instance really helped a lot in accelerate the training process. Get to set up the most up-to-date deep learning environment really benefits a lot.

8 Contributions

Data processing, literature review, AWS setup, open resource searching, model tuning, reporting and video are done by myself.

9 Github repo

https://github.com/Macavity77/cs230-hrcc (some reference repos can be found in the readme)

References

[1] CASIA offline/online character and text database. http://www.nlpr.ia.ac.cn/databases/handwriting/Home.html

[2] Xuefeng Xiao, Lianwen Jin, Yafeng Yang, Weixin Yang, Jun Sun, Tianhai Chang etc. Building Fast and Compact Convolutional Neural Networks for Offline Handwritten Chinese Character Recognition. arXiv:1702.07975.

[3] Zhang Y Deep convolutional network for handwritten chinese character recognition. Computer Sci- ence Department, Stanford University For further references see Deep Convolutional Network for Handwritten Chinese Character Recognition or go here: http://yuhao.im/files/ZhangCNNChar:pdf

[4] Z. Zhong, L. Jin, Z. Xie, High performance offline handwritten chinese character recognition us- ing googlenet and directional feature maps, in: Proceedings of International Conference on Document Analysis and Recognition (ICDAR), 2015, pp. 846–850.

[5] https://github.com/lucaskjaero/PyCasia/blob/master/pycasia/CASIA.py