Traditional Chinese Calligraphy Recognition Using Convolution Neural Network

Houze Liu
College of Engineering
Northeastern University
liu.hou@husky.neu.edu

Abstract- *Our goal of this project is to recognize Traditional Chinese Calligraphy in different styles. We will train Convolutional Neural Network on 100 labeled classes, each of which represents one kind of Chinese character. For every kind of character, 400 images are provided. We preprocessed our images all into one uniform size (128x128), and tried different net configurations. We changed the number of parameters of fully - connected layers', and constructed CNN models with different convolution layers. All models were compared their performance on test-set with top-1 accuracy. The final result was top-5 accuracy of prediction on validation-set.*

I. Introduction

Traditional Chinese calligraphy (TCC) is an essential part of Chinese traditional art and culture. The earliest TCC work can be dated back to 11 century BC. Along the history of Chinese culture, calligraphy has been developed far beyond clerical purpose; being able to write and judge calligraphy work was regarded as an evidence of being educated, intellectual and even privileged. Nowadays calligraphy has become a pure format for art performing. Recognizing calligraphy in art font is hard for human, not mention to a computer. Prior to our project, many approaches have been introduced. Most are based on certain feature extraction and K-nearest neighbor technique. However, CNN has been widely used on hand-written character recognition. Therefore we want to explore the performance of CNN on TCC recognition. Unfortunately, even though TCC is one special type of hand-written characters, it doesn't make recognizing TCC styles and recognizing hand-written characters more similar tasks. Most TCC are written by traditional Chinese brushes, which make the strokes much thicker than normal hand-written characters and as a result store more shape information. What's more, same Chinese character written in simplified form and in traditional form respectively can vary dramatically in body structure and layout. Last but not least, some TCC styles were created for aesthetic purpose, making characters written in those styles very hard to recognize even for natives.

Figure 1. Two characters in different forms

Compare 4 different calligraphies in one block, we see that they have totally different shapes, styles even body structures yet still are derivations of a single character that represent same meaning (All in left block mean 'East', right block mean 'Frigid').
In one block, the upper left is Seal style, upper right is Semi-cursive style, lower left is much simplified form and lower right is traditional form. Sometimes a calligraphy work in traditional form and then another in simplified form can be so distinctive that it becomes difficult to tell if the writer meant to 'drawn' the same character, unless the audience is familiar with Chinese characters in both forms.

II. Related Work

In our literature search, we found that there are recent works, which implement CNN technique on Chinese character recognition. Li [1] 's work was on recognizing TCC styles using different CNN models, which were similar to what we did. However, in our project we have 100 classes instead of only 5 styles. We also tried different combination of hyper-parameter choices.

In TCC recognition, most related works were focused on extracting features manually, like HOG feature and corner point feature. K nearest neighbor is widely used as classifier to solve recognition problem. We didn't find examples of using CNN on TCC recognition

III. Methods

We use architecture of CNN based on VGGnet model[4]. The goal of our project is to investigate what kinds of change in net architecture can effect model performance and how, so we divided our experiments into two parts. In the first part, we tried to find the best FC-layer set-up.

In part one, all models are kept the same convolutional layers and only vary in FC-layers as in Table 2. We added the number of FC-layers or doubled the number of parameters of every layer each time and compare results based on top-1 accuracy prediction on test-set. Generally, our models have 7 configurations, and we denote them from A to G. Within each, we also changed the number of parameters of every layer by doubling the number. Since the purpose of our experiment was not only to find the best model, but also to reveal more information about the relationship between layers and performance, we designed this workflow so that the results can indicate the next step of experiment. And apart from the question of how many, we also inquired the question of better arrangement. Thus in this part, our goal is not to get highest accuracy, but to find factors that might help increase it. Our

ultimate FC-layers set-up is different from every model presented on Table 2.

In the second part, we used a baseline L8 model, which has 3 convolutional layers consistent to part one. Its FC-layers are what we thought best suitable according to results of part one.

Table 1. CNN configuration of 8 different models

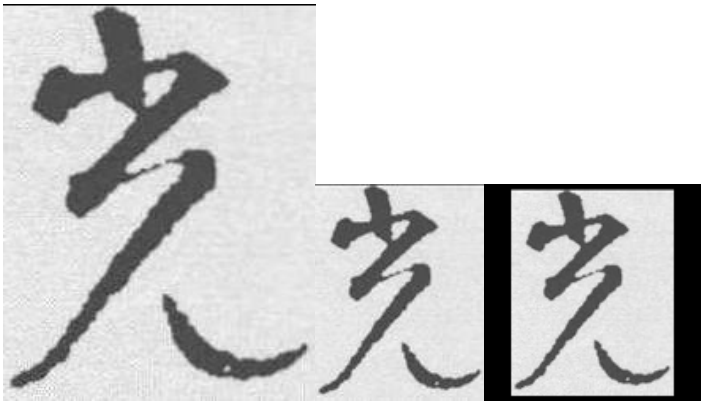
CNN Configuration							
L8	L9	L9+	L11	L11+	L13	L15	L15+
Input Images: 1x128x128							
32	32	64	32	32	32	32	32
			32		32	32	32
Maxpooling (2x2)							
64	64	128	64	64	64	64	64
			64		64	64	64
Maxpooling (2x2)							
128	128	256	128	128	128	128	128
				128	128	128	128
Maxpooling (2x2)							
	256	512	256	256	256	256	256
				256	256	256	256
						256	**Maxpooling (2x2)**
						256	256
							256
Maxpooling (2x2)							
FC-4000							
FC-2048							
FC-1024							
FC-1024							
FC-100							
Softmax							

We increase one more convolutional layer in L9 and L9+ models. The difference between L9 and L9+ is that we increase the number of channels in L9+ model. In L9 and L9+ model, we add two more convolutional layers between non-linearity and max-pooling. As is described by [2], we stack two convolutional layers with 3x3 filters. The stacking of two 3x3 convolutional layers is equivalent to one 5x5 convolutional layers in terms of the effective receptive field, but with fewer number of parameters and more non-linearity activation layers.

Between L11, L11+ and L13, we want to test putting additional layers in the front versus in the back of the convolutional layers. We added 2 more convolutional layers in L15 and L15+ models than in L13. We also used Batch-normalization to help converge.

IV. Dataset and Pre-processing

We resized every image from its original size to 96 as the size of shorter side, remaining longer side scaled proportional to shorter side. Then we adjust images to size of 128x128, padding the sides less than and cropping the sides larger than 128. So we get all images uniformly 128x128 so keep as much shape information as possible.



Size: 251x198 Size: 122x96 Size: 128x128

Figure 2. Pre-processing: Raw Image, Resized, Center Cropped

In our dataset, we have 100 different files, each of which contains 400 images of the same kind (40,000 in total). We divided our dataset into training set, test-set and validation set with proportion 8 : 1 : 1, which means 38000 for training, 1000 for test and 1000 for validation. All images are in gray-scale, and their sizes vary from (28, 43) at smallest to (1684, 2280) for largest image. So the first problem is to uniform image sizes in order to feed them to CNN. Our first try was spatial pyramid pooling, which doesn't require same input size. But we didn't have enough memory to run our models implemented with this technique at large scale. So we turned to crop our images to a certain square size. We tried 96x96, only to find that too many row images were deleted from some critical parts so that even human could not recognize. So we used 128x128. And another challenge was that some of our row images are wrong labeled, or heavily corrupted. Some of them are attached with seals, stamps or signature. However, we didn't deal with those circumstances because we wanted to see how well CNN could handle them.

Figure 3. Dataset visualization

V. EXPERIMENTS

We decided to first set up FC layers because the majority portion of memory was consumed by parameter storage. And FC layer, especially the first FC layer, contained much more parameters than others. And we thought that more FC layers could lead to better performance. So if we can find a appropriate way to set up FC layers, we will save many parameters so that we might finally configure more complex and deep models to try different possibilities.

Table 2. FC set-up based on same convolutional set-up

Set-up FC-layers		
A	FC **256** ->FC **128**	70.9
	FC **512** ->FC **256**	71.7
	FC **1024**->FC **512**	75.8
B	FC **256**	71.5
	FC **512**	72.3
	FC **1024**	73.3
C	FC **2000**->FC **1024**	75.9
	FC **4000**->FC **2048**	79.4
D	FC **512** ->FC **512** ->FC **256**	71.7
	FC **1024**->FC **1024**->FC **512**	77.4
E	FC **512** ->FC **256** ->FC **256**	74.6
	FC **1024**->FC **512** ->FC **512**	79.2
F	FC **512** ->FC **512** ->FC **256** ->FC **256**	73.7
	FC **1024**->FC **1024**->FC **512** ->FC **512**	76.2
G	FC **256** ->FC **512**	73.9
	FC **512** ->FC **1024**	75.3

Part A. All models in this part were trained 30 epochs using SGD with learning rate = 0.01, momentum = 0.9. ReLu was used as activation function. No special initialization, neither decay.

1.Compare A2, A3, B2, we can see that adding FC 1024 at front is better than adding FC 256 at back position.

2.Compare D1=FC 512 -> FC 512 -> FC 256
E1= FC 512 -> FC 256 -> FC256
D2= FC 1024 -> FC 1024 -> FC 512
E2= FC 1024 -> FC 512 -> FC 512

Since E1 > D1 and E2 > D2, we should avoid the same layer at very front position. If we want to add a clone layer, we should put it at back.

3.Compare D1 < F1; D2 > F2: adding layer at back is good when parameters are few, not good when many.

4.Compare A2, A3 and G: sequence doesn't influence much. The accuracy increase from A2 to G1 might be due to no sufficient parameters at back position.

5.Compare B: adding parameter is good for one layer.

6.Compare D1, E1; D2, E2: reducing middle layer parameters helps get higher accuracy.

7.Compare E, F: adding same layers at front position doesn't help

8. Compare all: adding parameters at same layer respectively is always good

We didn't use any techniques to initialize parameters. So due to randomness in initialization, every time experiment result may vary. Accuracy gap within 1 will be regarded as not obvious.

To conclude, ways to improve include: more parameters at first FC-layer, no clone layer at front, more parameters in total, reverse pyramid of parameter numbers, more FC-layers.

In order to explore more structural advantages, we decided to combine C2 and E2 – two models with best performance – together. Following previous observations, we came to a model that has more parameters at front position, more layers while avoid having copy layers at first and two front position:

FC 4000 -> FC 2048 -> FC 1024 -> FC 1024

It has a very stable accuracy of 78.8%. We also tried to change parameters in each layers. We found that only either adding parameter at first layer or reducing parameters at last layer worked: only to keep accuracy from dropping. It might because that the stability comes from its structural benefits. Considering that adding parameters in the first layer will introduce more parameters in total, we decided to undo unnecessary changes and use it.

Part B. In our test on models from L8 to L15+, we found that adding convolution layers and adding more channels are both useful in terms of higher accuracy on test-set.

However, putting the extra layers at front is slightly better than at back, as the results of L11 and L11+ show. In our case, since stacking layers means that wider receptive fields are functioning, we can turn our observation into another form of conclusion: adding one convolutional at back has most significant improvement. Doubling channels improves slightly. Using broader receptive field at front is better than doing so at back.

In L15+, one more max-pooling layer was inserted in last four convolutional layers down-sampling features. Because if down-sampling has no negative influence on model performance, more max-pooling layers are better in terms of reducing size of final convolutional feature map, and less parameter, as result.

Another discovery was that model L13, L15 and L15+ didn't converge under the circumstances of optimization we chose to use. In this case, we hoped that those models could provide larger capacity on our data features if they could converge. So we implemented batch-normalization on those models.

The batch normalization layer performs normalization for each training mini-batch. In training process, it computes the mean and variance over the mini-batch input and then normalizes the input. And the scale and shift parameters can be learned during the training process. The equations are shown below:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{1}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \tag{2}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{3}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \tag{4}$$

According to Ioffe and Szegedy[3], batch normalization is able to stabilize the learning process, and allow using of higher learning rate.

When we added batch-normalization after every convolutional layer, the model began to converge at a speed even greater than models built with simpler structure and less layers. In Figure 2,we see that L13(represented by silver line) is the first model that plateaus. The flat lines indicate that complex models like

L11, L11, L15, L15+ even showed no sign of converging along whole training process.

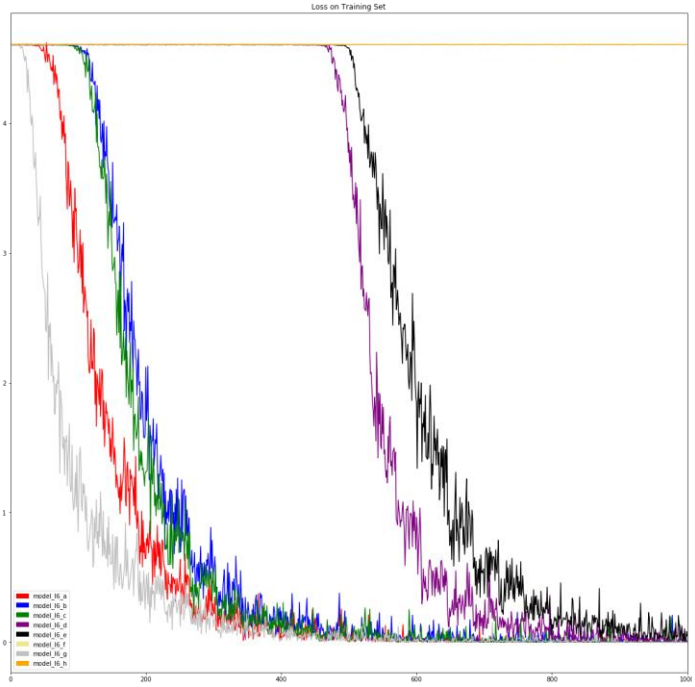


Figure 2. Plot losses of all models trained 50 epochs.

Since L15+ with batch-normalization has less accuracy than L15 with batch-normalization, the idea of implementing one more max-pooling layer was refuted.

L15 with batch-normalization (abbreviate as L15bn) outperformed all other models in our experiment so far. Our only doubt was if we could get higher accuracy by adding more channels to L15bn. Because of memory restriction (60G), doubling all channels was the most we could do to L15bn while keeping FC layers unchanged. The new model had slightly better performance at average than L15bn while slightly worse at its highest accuracy than that of L15bn. However, considering that it is much more time consuming to train a complex model with more epochs and each epoch takes more time to finish, we eventually settled down L15bn.

L15bn best performance:

```
Train Epoch: 72 [35000/38000 (92.105%)] Loss: 0.000010  Accuracy: 1.000
Train Epoch: 72 [36000/38000 (94.737%)] Loss: 0.000004  Accuracy: 1.000
Train Epoch: 72 [37000/38000 (97.368%)] Loss: 0.000202  Accuracy: 1.000
accuracy on test set: 91.900% 919
Train Epoch: 73 [0/38000 (0.000%)]      Loss: 0.020242  Accuracy: 0.990
Train Epoch: 73 [1000/38000 (2.632%)]   Loss: 0.000009  Accuracy: 1.000
Train Epoch: 73 [2000/38000 (5.263%)]   Loss: 0.000005  Accuracy: 1.000
```

L15bn (channel doubled) best performance:

```
Train Epoch: 77 [36000/38000 (94.737%)] Loss: 0.000004  Accuracy: 1.000
Train Epoch: 77 [37000/38000 (97.368%)] Loss: 0.000009  Accuracy: 1.000
accuracy on test set: 91.600% 916
Train Epoch: 78 [0/38000 (0.000%)]      Loss: 0.018183  Accuracy: 0.990
Train Epoch: 78 [1000/38000 (2.632%)]   Loss: 0.000007  Accuracy: 1.000
```

Figure 3. Two L15bn models trained with 80 epochs.

In the next step, we tried Xavier initialization, dropout technique, tuning learning rate and epochs to find the best model configure. We found that the model with no any optimization had best both top-1 accuracy and top-5 accuracy on validation-set. Top-1: 91.9% and Top-5: 97.5%

Code and Documentation:
https://github.com/HouHouHouHouHouHou/7390

VI. CONCLUSION

In this project, we explored the performance of CNN models on TCC recognition. We constructed different models with different numbers of convolutional layers and different numbers of fully connected layers. And we found that in our case, the number of FC layer and that of parameters both have influence on accuracy. We also observed that using certain structure of FC layers could achieve the most stable performance. When it comes to convolutional layer, we found that more channels meant higher accuracy. It might because that the more filters, the more parameters, which means that the model is capable of fitting more complex feature space and higher dimensional data. As a result, it can reveal more information that helps it better classify images. But too many layers sometimes can kill the gradient. On the other hand, we inferred that more layers could improve model performance. In order to verify that, we implemented batch-normalization to help stuck models converge, and finally achieved the best result in this project. Nevertheless, we still didn't know if this improvement was due to adding more layers, or due to batch-normalization itself, and if so how the proportion of effect should be taken to either side. In our future work, we might need to add batch-normalization to all models and see if their performance would improve significantly. Another work was to fine-tune our L15bn model. We might be able to achieve higher accuracy using this model by reducing learning rate or applying decay schedule. And we would do more experiments for each models and average the results so that our evidence will be more convincing. For the same reason, we should've used normalized initialization values. Last but not least, data preprocess could be better done, for example, we can use spatial pyramid pooling technique to handle different input size so that we can retain more row image information.

REFERENCES

[1] Boqi Li. "Convolution Neural Network for Traditional Chinese Calligraphy Recognition," CS231N course project

[2] Simonyan, Karen, and Andrew Zisserman."Very deep convolutional networks for large-scale image recognition," International Conference on Learning Representations, 2015.

[3] Ioffe, Sergey, and Christian Szegedy."Batch normalization: Accelerating deep network training by reducing internal covariate shift," arXiv preprint arXiv:1502.03167 (2015).

[4] He, Kaiming, et al."Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," Proceedings of the IEEE International Conference on Computer Vision, 2015.

[5] Pengyuan Lyu, Xiang Bai, Cong Yao, et al."Auto-Encoder Guided GAN for Chinese Calligraphy Synthesis," arXiv preprint arXiv:1706.08789v1, 27 Jun 2017.

[6] TinyMind.cn database."TinyMind first TCC recognition competition," http://www.tinymind.cn/competitions/41?from=blog