CS230

Recognition of East Asian language characters

Yi-Ting Chen
Department of Applied Physics
Stanford University
yitchen@stanford.edu

Po-Nan Li*
Department of Electrical Engineering
Stanford University
liponan@stanford.edu

Abstract

This project addresses the image classification of East Asian characters by using a convolutional neural network. The trained network reaches the classification accuracy of 95.62%, and up to 99.75% if Traditional and Simplified Chinese are seen as a single label. We further performed feature visualization, deconvolution and class model analyses, which reveal that our network correctly learned most of the frequent characters that can distinguish the four systems under consideration.

1 Introduction

Online translation tools have greatly changed the way people learn and use foreign languages. In particular, augmented reality techniques such as "Word Lens" have made the translation even handy by allowing users to capture the foreign words with their mobile phones so they don't need to know how to type them in. Such applications, however, usually require users to choose the destination language as well as the source language, which users probably don't know. In this project, we aim to train a convolutional neural network (CNN) [1] model that can identify the language of words or characters coming from an image. Specifically, we are interested in the classification of four common writing systems in East Asia, i.e. Traditional and Simplified Chinese (abbreviated as TC and SC, respectively), Japanese and Korean.

The four writing systems considered in this project inherently share many common features. For example, Japanese *Kanji* and Korean *Hanja* are greatly adopted from Traditional Chinese characters, although Japanese also has two other components in its writing system, namely *Hiragana* and *Katakana* and Korean's another component, namely *Hangul*, is largely used over *Hanja*. Additionally, Simplified Chinese, one of the contemporary Chinese writing systems used in Chinese community, also shares many common features from the stroke to character level. These factors combined makes the classification of these East Asian writing systems an intriguing problem: can the network detect the subtle difference in the Pan-Chinese writing systems? At a higher level, we would also like to know if the CNN can learn the common building components of East Asian languages and use them as convolutional features.

2 Related work

Two common approaches for image-based language identification are currently utilized. The first one involves image segmentation of characters followed by text-based language identification [2]. The second one starts with feature extraction of images followed by clustering algorithm [3, 4, 5]. Among existing image-based works, little has been done with deep learning models, although there

*http://ponan.li/

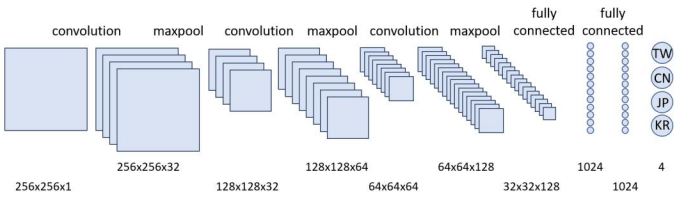

Figure 1: Architecture of the 5-layer CNN considered in this project.

A

DEV Accuracy		keep prob 0.5	keep prob 0.9
model	Small	94.73%	92.89%
model	Normal	95.28%	94.21%

B

DEV Variance		keep prob 0.5	keep prob 0.9
model	Small	3.4%	5.8%
model	Normal	4.0%	5.4%

Figure 2: Dev set (A) accuracy and (B) variance with different hyperparameters.

are a number of CNN models for text-based language identification using features extracted from text [6, 7, 8]. Different from these methods, in this project, our implementation will be purely image-based using CNN. Unlike Ref. [7, 8], our features will be directly extracted from images, which is an area that was less explored.

3 Dataset and Features

We built a data crawler that collects Traditional Chineses (labelled as TW), Simplified Chinese (CN), Japanese (JP) and Korean (KR) versions of transcripts of all public videos on `TED.com` as our corpus. The collected texts are then split into sentences, each of which is converted to a 256x256 grayscale PNG file with random typeface, font size and random noise. We ensured each generated picture contains at least five characters. The whole dataset totals 2.5 millions pictures with 25% of each writing system and is then split into training, dev and test sets in a 0.90:0.05:0.05 fashion.

4 Methods

We built a CNN and developed a collection of tools for data preprocessing and post-analysis. The whole pipeline is developed with Tensorflow [9]. Figure 1 shows the architecture of our CNN, where 3 convolutional and maxpool layers are used to capture the structures of characters on the training images, and two fully-connected layers and a soft-max is used to predict the language of the characters on the image. The loss function is the cross entropy, defined by $L = -\sum_{data} y \log(\tilde{y})$, where y is the label and $\tilde{y}$ is the prediction from our CNN model. The Adam optimizer was used to train the model.

5 Results

The neural network was trained for 200K iterations with minibatch size 128 on a GPU-enabled AWS instance. With the above described model, we achieve the accuracy of 99.3% and 95.6% on train set and dev set, respectively. As the bias is decently good, we then aimed at reducing the variance by tuning hyperparameters.

5.1 Hyperparameter turning

We use two hyperparameters to adjust the extent of regularization. 1) Model size: Normal size as shown in Figure 1, and small size where all filters are cut half. 2) Dropout probability: Keep prob = 0.5 and 0.9. As shown in Figure 2, low keep prob enforces regularization to the model so that all models with keep prob=0.5 consistently have lower variance. Among the model with low keep prob, the model with normal size has better dev set accuracy. As a result, we chose the model with keep prob=0.5 and normal size as the best performance model.

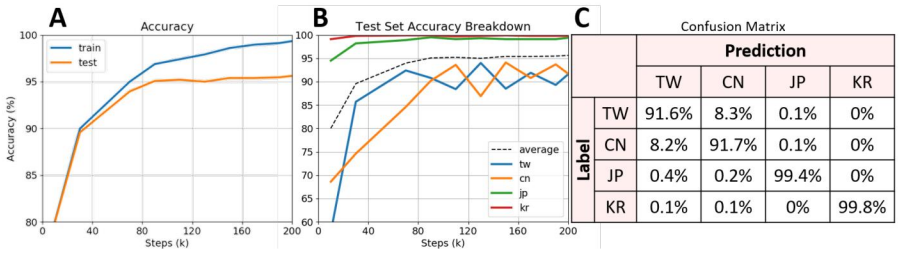

Figure 3: (A) The accuracy on train and test set. (B) The accuracy of the individual languages on test set. (C) The confusion matrix on test set.

5.2 Accuracy analysis

The best model scored accuracy of 99.34% and 95.62% on the train and test set, respectively (Figure 3(A)). More insightful details are revealed as we break down the accuracy of the test into individual languages (Figure 3(B)). It is clear that, while JP and KR have accuracy above 99%, the TW and cn are only around 90%. This relatively low accuracy in TW and CN can be explained in the confusion matrix (Figure 3(C)). Most of the confusions happen only between TW and CN. For example, in the confusion matrix, about 8% of tw data are are wrongly classified as CN, while the fraction that are predicted as KR is less than 0.1%.

The error between TW and CN is probably due to the fact that traditional and simplified Chinese share a lot of similar or even identical characters. On top of that, the strokes in characters of TW and CN have similar style, making it harder to distinguish them. More details about this issue will be discussed in a following section. TW and CN have high similarity. People who use TW can usually understand CN, and vise versa. So one aspect of analysis is to combine them as the same label. In this case, the accuracies are 99.98% and 99.75% on training and test sets, and the variance is 0.23%.

5.3 What did the classifier learn?

To reveal what features has the network learned, we performed various analysis approaches, including feature visualization, deconvolution and class model generation.

5.3.1 Image patch visualization

After the CNN is trained, we fed all images in the datasets to the network and recorded the top nine images that best triggered each activation layer in the network and the location of the neuron with highest activation. If the hottest neuron is found at (u, v) of the l-th layer's w-th map, the image patch that is responsible for this activation can be found at the box between the corners (x_L, y_T) and (x_R, y_B), where $x_L = u \prod_i^l s_i$, s_i being the number of stride at the i-th layer, and $x_R = x_L + \sum_i^l \left[k_i \prod_{j=i+1}^l s_j \right] + 1$, k_i being the filter size at the i-th layer; y_T and y_B can be derived in the same way.

Figures 4 (A-C) show image patches that best activated the neurons in layer 1 to 3, respectively. Top nine examples were presented for each filter. For example, Figure 4(D) shows nine examples of image patch that would greatly trigger one of the 32 filters in the first layer. Interestingly, we found the CNN tends to learn the shapes of frequent words in the training corpus. For example, as illustrated in Figures 4(F) and (G), the character representing plurality of personal pronoun (such as we, you and they) in both TC and SC were learned by the CNN to, ultimately, distinguish TW and CN.

5.3.2 Deconvolution

We further implemented a deconv-net to visualize the filters in the CNN. We recorded the locations of the maxima when performing the maxpooling in the forward propagation, then preserve the neuron with highest activation in the third layer, and perform un-activation, un-maxpooling and deconvolution, as instructed in Ref. [10]. Figure 5 shows the deconvolution of two representative examples. In Figure 5(A), for example, bright pixels in the deconvolved layer[1] indicate the locations

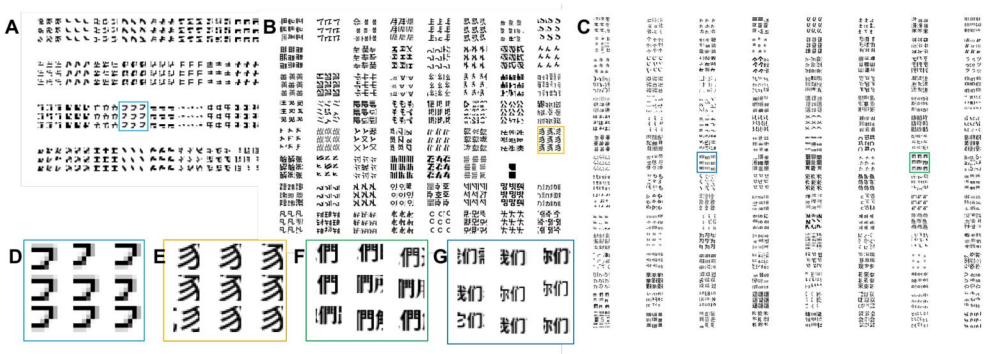

Figure 4: Image patches that best triggered the neurons in (A) 1st, (B) 2nd and (C) 3rd layer. Note that some cells in (B) are blank, presumably suggesting that some filters were never fired. High resolution images are available on the GitHub repository. (D) A representative sample picked from (A) shows nine image patches that maximize the activation of one of 1st layer's 32 filters. (E) A representative sample picked from (B) shows nine image patches that maximize the activation of one of 2nd layer's 64 filters. (F,G) Two representative samples picked from (C) show that the CNN recognizes both the Traditional and Simplified versions of the same frequent words to distinguish them.

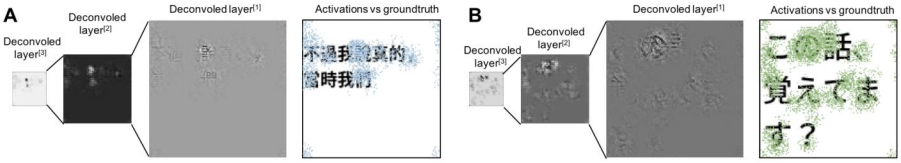

Figure 5: Deconvolution-net visualizes the components that triggered the activation. (A) Representative example of deconvolved Traditional Chinese sample. Keywords that distinguish TC against SC fired strong activation. (B) Representative example of deconvolved Japanese sample. Hiragana characters, one of three components in Japanese writing system, fired strong activation.

that trigger the activations in the following layers, which are superpositioned with the groundtruth for visualization.

5.3.3 Class model generation by gradient ascent

Gradient ascent is another method to visualize the features learned by the model [11]. It starts with a randomly initialized image, which then is updated to maximize a score function constrained by the $L1$ norm. Figures 6(A) and (B) show two examples of gradient ascent visualization on the first layer of our CNN. It is clear that their corresponding filters are sensitive to diagonal and horizontal features, respectively. Applied in the output layer, the gradient ascent is able to transform an image from one class to another. it starts with a data image, which then is updated to maximize the score of a language. As an example in Figure 6(C), an image is classified as KR, and end up being classified as TW after gradient ascent. Instead of transforming to real characters in TW, it adds background to image. This might suggest that being able to deal with adversary and training on practical noisy are necessary for a more robust classifier.

5.4 Real-time recognition

To help participants of the CS230 poster session better understand our work, we developed a program that recognized the characters in the image that captured by the webcam in real-time, as shown in Figure 7. Enabled by OpenCV [12], the program constantly fetches the image from the webcam, converts to grayscale and feeds it the trained network. This program is highly responsive and runs smoothly on a laptop without a GPU thanks to the portablility of TensorFlow's frozen graph. Interestingly, this real-time implementation predicts on webcam images decently, despite that we never trained the network with camera data.

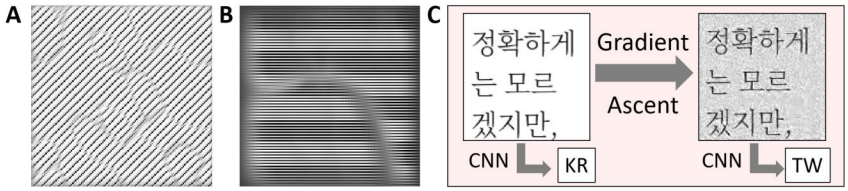

Figure 6: (A,B) Examples of generated images that maximize the activation of two filters in layer 1 of CNN. (C) Use gradient to transform an input labelled as KR to the one recognized as TW the by the CNN.

Figure 7: (A) Graphic user interface of the real-time reorganization program, where a print-out of the Korean version Wikipedia page of "Machine Learning" is being tested. Box with white border in the center is the region of interest. Red label indicates the prediction and the black texts display the breakdown of catalogical probability. (B-D) Snapshots of the program being tested with Wikipedia page of "Machine Learning" in (B) Japanese, (C) Simplified Chinese and (D) Traditional Chinese.

6 Discussion

As discussed in Sec. 5.2, the classification error is largely due to the confusion between TW and CN, which intrigues us to find out that how did the CNN distinguish TW and CN. Although a user of either TW or CN would agree that one is greatly different from the other, a large portion of characters that exist in both systems are, however, identically the same. Surprisingly, the word frequency statistics of each case shows that among the top 10 frequent characters, which collectively count 20% in the corpus, only two characters are different. Indeed, these "key words" that can be used to distinguish TW and CN are favored by the CNN and are responsible for the activations in the convolutional layers, as revealed in Sec. 5.3.1 and 5.3.2.

Japanese, on the other hand, is usually recognized by the frequent existence of *Hiragana* characters. Furthermore, although we cannot provide insights on how did the CNN recognize Korean, which it did nearly perfectly, we found worthy mentioning that English and other Latin characters, which occasionally appear in the corpus, never activated the CNN as they are irrelevant to the classification of CNN's interest.

7 Conclusion

We trained a five-layer CNN that can identify the writing system of the East Asian characters on a given image. The trained network reached accuracy of 95.62% on the test dataset and variance of 4%. Notably, our error analysis suggest that the classification error is largely due to the confusion between Traditional and Simplified Chinese. To understand how did the CNN recognize and distinguish the four writing systems, we performed various qualitative analyses, which consistently show that the trained CNN uses characters that are frequently used in each of the writing system as features for each of the convolutional layers. Result of class model image generation by gradient ascent, however, suggests that our model might be susceptible to noisy or adversarial inputs. Future works might include more Asian and Latin languages and more realistic training data such as those from cameras.

Acknowledgment

The authors are grateful for the inputs from Hsin-Hung Yeh, a PhD candidate at Department of East Asian Languages and Cultures.

Code

Code of this project is publicly available at `https://github.com/leeneil/ealc`.

Contributions

Both authors conceived the research topic. Y.-T. Chen built the deep learning pipeline, performed hyperparameter turning, managed computing resources and contributed to the gradient ascent analysis. P.-N. Li developed the data crawler and generator, and contributed to the visualization of image patches, deconvolution and real-time recognition. Both Chen and Li trained the net, evaluated the results and wrote the report.

References

[1] Yann LeCun, Léeon Bottou, Yoshua Bengio, and Patrick Haffner, "GradientBased Learning Applied to Document Recognition," Proc. IEEE (1998).

[2] P. Barlas, D. Hebert, C. Chatelain, S. Adam and T. Paquet, "Language Identification in Document Images," Journal of Imaging Science and Technology (2016)

[3] G.S. Peake and T.N.Tan, "Script and language identification from document images," Document Image Analysis (1997)

[4] Ying-Ho Liu, Chin-Chin Lin and Fu Chang, "Language identification of character images using machine learning techniques," Document Analysis and Recognition (2005)

[5] Shijian Lu, Chew Lim Tan and Weihua Huang, "Language Identification in Degraded and Distorted Document Images," Document Analysis Systems VII, pp 232-242 (2006)

[6] C. Cireşan, Ueli Meier and Jürgen Schmidhuber, "Transfer learning for Latin and Chinese characters with Deep Neural Networks," WCCI 2012 IEEE World Congress on Computational Intelligence.

[7] Yoon Kim, "Convolutional Neural Networks for Sentence Classification" (2014).

[8] Xiang Zhang, Junbo Zhao and Yann LeCun, "Character-level Convolutional Networks for Text Classification," NIPS 2015.

[9] Martín Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," Software available from `tensorflow.org` (2015).

[10] Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," arXiv:1311.2901 (2013).

[11] Karen Simonyan, Andrea Vedaldi and Andrew Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," arXiv:1312.6034 (2014).

[12] Itseez, "Open Source Computer Vision Library," `https://github.com/itseez/opencv` (2015).