Handwritten Korean Character Recognition Using Convolutional Neural Network

Joonil Ahn
Department of Mechanical and Industrial Engineering
University of Toronto
joonil.ahn@mail.utoronto.ca

Abstract

In this work we use VGG-19 [1] pre-trained model to classify 2.1 million images of handwritten Korean characters into 2,350 labels. The pre-trained model was trained on ImageNet for 1,000 number of classes. We fine-tuned the weights in 3 fully connected layers by freezing weights in the convolution layers of the model. We trained the model for two cases, 256 classes and 2,350 classes, to show how the performance changes with regards to the number of classes. We extracted images for 256 most frequently used characters for the first model and the entire images for the 2,350 classes case. We achieved 91.05% accuracy on test set for 256 classes and 84.57% accuracy on 2,350 classes case.

1 Introduction

Handwritten Korean character (Hangul) recognition is a still difficult problem to solve due to its complex structure and variations. Korean words can are divided into blocks of characters. Each character consists of an initial consonant cluster, a medial vowel and optionally a final consonant cluster. In other words, a several consonants and vowels combine together to form a single character, hence recognizing Hangul character is more difficult than English character. There are 11,172 possible Korean syllables with 19 initials, 21 medials and 27 finals. Figure 1 shows typical structures of Hangul characters.

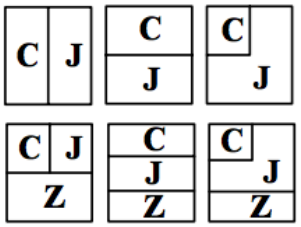


Figure 1: Blocked Structure of Korean Hangul (C: Initial, J: Medial and Z: Final)[2]

If we want to discriminate all the possible letters in Korean, it becomes 11,172 class classification problem. However, it is difficult for researcher to collect sufficient number of data for the 11,172 classes. Since most of them are not used in modern Korean words, we only consider 2,350 characters in this work which are the characters can be represented by EUC-KR encoding. Even 2,350 number is not small but we can tackle the problem with state-of-the-art CNN models.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

2 Related Work

Recognizing handwritten characters have been widely studied by many computer scientists. LeCun et al. proposed LeNet-5, a convolutional neural network, to discriminate MNIST handwritten digits dataset[3]. Although it significantly affected modern CNN models, the architecture is not sufficiently deep network to be applied to complex syllables like oriental syllables.

However, Korean handwritten recognition problem was still hard to solve due to lack of data and structural complexity. In general, Korean handwritten recognition can be categorized in to two groups; structural method and statistical method. In the early days, statistical methods were widely used to classify handwritten Korean [4][5]. Kim et al. proposed a model based on hierarchical random graph representation[6]. The model learns its parameters of the hierarchical graph using EM algorithm and embedded training technique.

As deep neural networks boosted image classification performances in many domains, however, many researchers also tried to apply CNNs to handwritten Korean recognition. Kim et al. built Korean characters recognizer using CNN with 4 convolutional layers and 2 fully-connected layers[7]. They used SERI95a (520 classes) and PE92 (2,350 classes) datasets which contains approximately 1,000 images/class and 100 images/class, respectively. Their model achieved a recognition rate of 95.96% on SERI95a dataset, and 92.92% on PE92.

Despite the descent result, their work contains two major problems: 1) lack of data per class and 2) lack of test examples. For example, PE92 contains only 100 images per character class that is not sufficient number of data for deep learning. They also used 90% of data for the training and only 10% of data for the test.

3 Comparison

We will compare some different key points in terms of dataset and algorithms between Kim el al.'s model([7]) and our model.

1. Dataset
   Kim et al.[7] used PE92 and SERI95a datasets for their paper. The datasets were created in 1992 and 1995 respectively, hence resolution of the images are 64×64 which is not sufficient as input size for deeper networks.
   Instead, we use 2,133,694 number of handwritten Korean character images provided by EIRIC, Chung-Ang University, South Korea[8]. There are approximately 900 number of data for each character class, and 2,350 classes in total.

2. Preprocessing
   Images from SERI95a and PE92 seem clear and consistent while EIRIC's dataset we used in this work contain lots of noises in the image files. Some of EIRIC images have black vertical lines or dot noises on background of images etc. Size of the images also vary. So the images need to be preprocessed, such as denoise, crop and resize, before we feed them into the model.

3. Fine tuning vs. Full training
   We used fine tuning strategy using pre-trained VGG-19 model to save time for training the model. We only update weights for last four fc-layers to tune the weights. Kim et al. used 4 conv-layers with max-pooling, 2 fc-layers while VGG-19 pre-trained model has 5 conv-layers with batch normalization and max-pooling and 4 fc-layers with dropout. Batch normalization and dropout layers might help generalize the model for unseen data.

   Kim et al. spent 26 days to complete 500 epochs for training their model, while we could achieve competitive result within 3 days. Personal computer was used for the training with the following specification.

   - CPU: Intel i7-6700K CPU @ 4.00GHz
   - GPU: NVIDIA GeForce GTX TITAN X (3,584 cuda cores and 12GB of memory)

4 Preprocessing Images

Our dataset includes over 2 million labeled handwritten Korean character images belongs to 2,350 kinds of characters (EIRIC, Chung-Ang University, South Korea[8]). The images are variant in resolutions while the model needs a consistent input size, 224 × 224. The handwritten characters also vary in thickness and style and some of the images contain some pixel noises created during scanning process. Therefore, we first need to remove the noise with median filter and crop out extra spaces outside the handwritten parts, then resize all the images to 224 × 224.

(a) Raw Images (b) Preprocessed Images

Figure 2: Samples from Dataset

5 Model

The model architecture, as shown in figure 2, is the CNN architecture of [1]. We do not modify the fundamentals of the original model but only change the last fc layer from 4,096×1,000 to 4,096×2,350 or 4,096×256 depending on the two different number of classes.

5.1 Convolution Layers

In l-th layer of a convolution layer, the following is iteratively computed in forward propagation.

$$x_{i,j,d}^{(l+1)} = \sum_{w=0}^{W} \sum_{h=0}^{H} (x_{i+w,j+h}^{(l)} \times M_{w,h,d}^{(l)} + b_d^{(l)}) \tag{1}$$

where i, j denote location of pixel, d denotes d-th depth of filter and W, H denote width and height of filter $M^{(l)}$ and b is a bias term.

Then we normalize the output of the convolution using batch normalization[9]. ReLU activation, $f(x) = max(0, x)$, is used as a non-linearlity for VGG-19.

VGG network repeats the computation above for 5 conv-layers with different filter depth sizes. Then the output from 5th conv-layer is flatten to 25,088 dimensional vector be passed to fc-layers. The only difference between VGG-19 and our model is the final output size. Since the number of classes is differed from ImageNet, our model's output should be 2,350. Let N be the number of data, n be n-th data point, K be the number of classes, k be k-th class, z_k^n be a score of kth class for nth data. Then the output layer computes the activation of the k class by the softmax function.

$$\mathbf{y}_k(\mathbf{x}^{(n)}) = \frac{e^{\mathbf{z}_k^{(n)}}}{\sum_j e^{\mathbf{z}_j}} \tag{2}$$

Then the cross-entropy loss can be derived by computing the negative log-likelihood for the following.

$$L = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_k^{(n)} \log y_k^{(n)}(\mathbf{x}^{(n)}) \tag{3}$$

We can update the gradients by implementing backpropagation through the fc-layers. We use adam optimizer for the gradient updates.

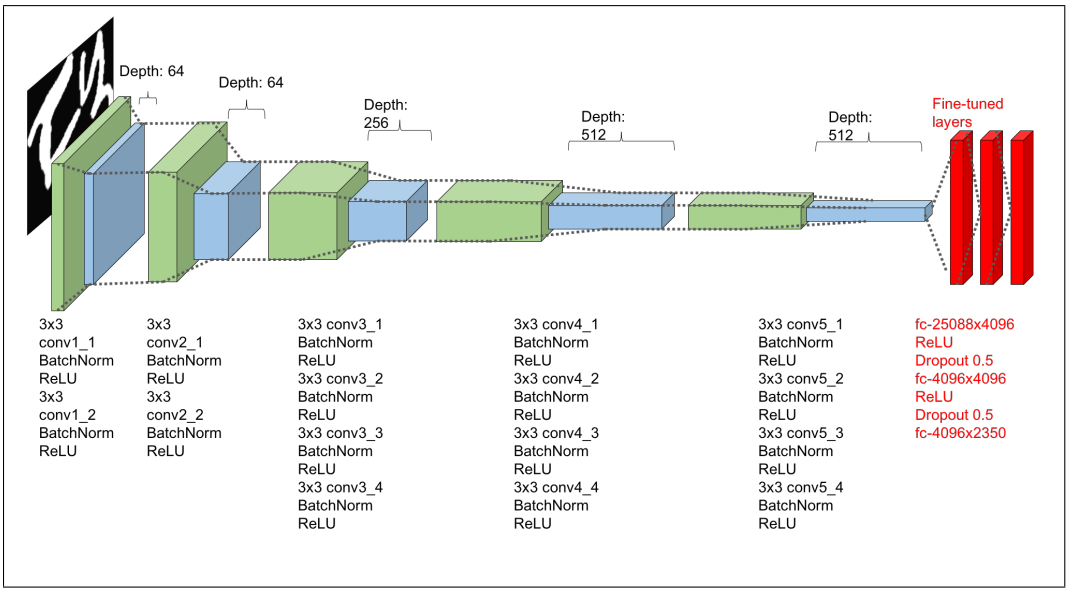


Figure 3: VGG-19 Architecture

6 Experiments

We need to preprocess the images to remove potential outliers and noises. The images were transformed by three steps: 1) denoise, 2) crop and 3) rescale.

In denoise step, we set a threshold to detect background noise and remove the noises. After reading an original image with gray scale, pixel values for the image are ranged from 0 to 255. Pixels for handwritten parts usually have low values (less than 100), hence we can consider large pixel values as noises. In this work, we set pixel values larger than 225 to be 255. Then, we rescale the range of the pixel values from [0, 255] to [0, 1] for numerical stability.

After denoising the images, we need to crop the images to ignore empty spaces of the images. By doing so, we might achieve better performance and faster convergence time during training. We simply find white pixels, whose value might larger than 0, and estimate coordinates for a rectangular which contains all the white pixels in the images. Finally, we should resize the size of the images to 224×224 because it it the input size for VGG-19 model.

Now, we can feed the images into the network to train the classifier. We load pre-trained VGG-19 model's weights from PyTorch module. The weights are used in forward propagation but not be updated in backpropagation. We start with 256 classes case to see how the fine tuning method is effective for this task and do the same task for 2,350 classes. 256 classes were adapted from 6,000 Korean words compiled by the National Institute of Korean Language[10].

We split the dataset into three non-overlapped subsets; 50% for train set, 20% for validation set and 20% for test set, respectively. At the end of each epoch, the algorithm checks how the model well generalizes for unseen data by computing validation error. To avoid overfitting, we keep a model showing the best validation accuracy. After completion of the entire training, we use the model to estimate accuracy for the test data. We use 128 batch size, 0.0001 initial learning rate, adam

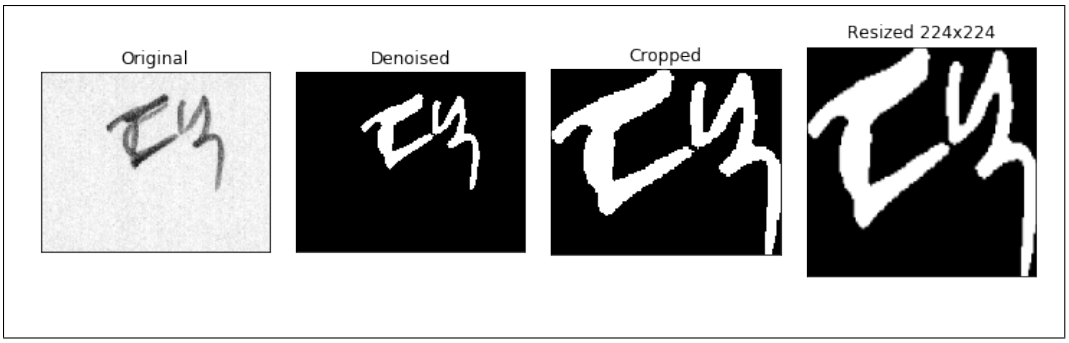


Figure 4: Image preprocessing

optimizer with 0.9 momentum for the training. The learning rate will decay by 0.95 every 5 epoch. We iteratively trained the examples for 20 epochs.

Table 1: Train / Validation / Test Splits

	Number of train data	Number of validation data	Number of test data
256 classes	141,389	47,129	47,129
2,350 classes	1,280,218	426,738	426,738

Table 2 shows train and test accuracy for 256 and 2,350 class cases we achieved from the experiments. We tried to avoid overfitting problem by selecting the model which showed the best accuracy for the validation set. We can see the accuracy for 2,350 classes is lower than that for 256 classes. As we predicted, discriminating a large number of classes is more difficult to solve.

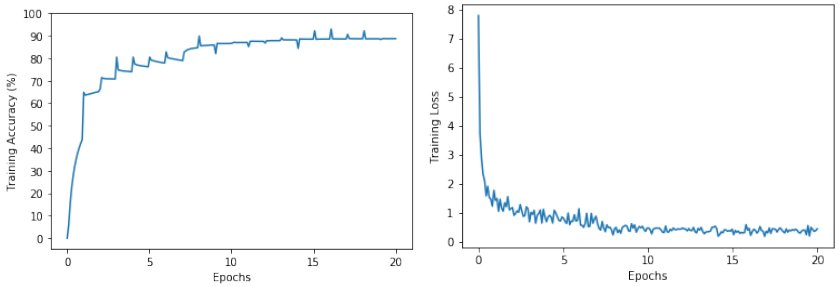
Table 2: Test Accuracy

	Train accuracy	Test accuracy
256 classes	99.95%	91.05 %
2,350 classes	96.69%	84.57%

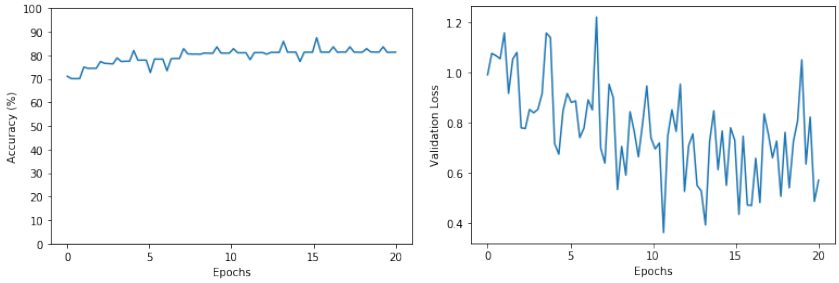
It is shown in figure 5 that training loss rapidly decreases until 5 epochs for 256 classes case. After 10 epochs, the loss does not noticeably change any more. This significantly fast convergence is possible thanks to the weights from the pre-trained model. It is known that the weight in conv-layers contain various informations about edges and shapes learned from many different images. By fine-tuning the fully connected layers of the model, weights start to capture some details of images about the Hangul characters.

7 Limitations

1. Too bold examples
   The model often fails to recognize a handwritten character if it is too bold. In preprocessing state, our algorithm crop an image and resize it to 224 $\times$ 224. If a character in the image is too hold, some parts of the character overlap each other then it is hard to be discriminated from similar characters.
2. Too similar characters
   Since character recognition model does not depend on a context of text, it is often confused about similar characters. For example, The most confused characters in 256 classes case are '별' and '벌'. Among 176 examples for '별', 24 examples were classified as '벌'. The second most confused characters are '담' and '당', recorded at 18 out of 192 examples for '담'. Even though you have no prior knowledge about Korean language, you can possible see the mistakes are due to similarity in shapes between those characters. Since this model's prediction is based on CNN, it only cares about the shapes of training images for the

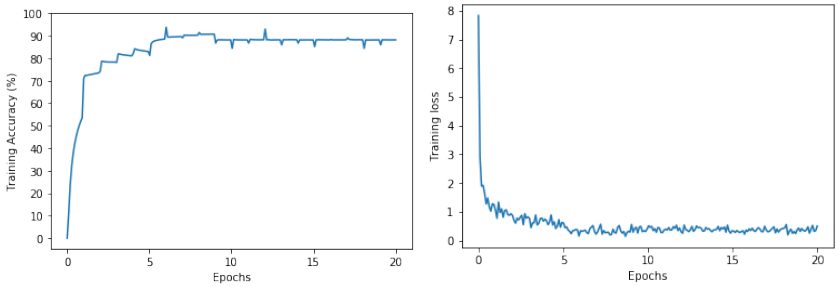


(a) Training Accuracy (b) Training Loss

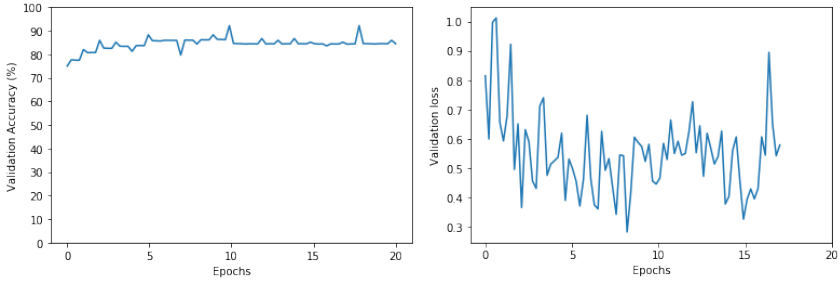


(c) Validation Accuracy (d) Validation Loss

Figure 5: Training/Validation Accuracy and Loss - 256 Classes



(a) Training Accuracy (b) Training Loss



(c) Validation Accuracy (d) Validation Loss

Figure 6: Training/Validation Accuracy and Loss - 2,350 Classes

corresponding labels. If we are building word classification model, we may have to consider order of characters for words.

How can we mitigate the confusions?

(a) Raw Image (b) Resized Image (c) Confusing Character

Figure 7: Example of Resizing Problems

1. Bold or thin examples: Preprocess them to have consistent thickness of pixels. Algorithms should be studied in future work.
2. Train models for consonants and vowels separately and combine together for inference. If we train models for the 19 consonants and 21 vowels, it will hugely save time for the training. However, we should be careful about orders of consonants and vowels because Korean characters consist of one or two consonants and one vowel. A first consonant is called the initial (onset), and a vowel is called the medial (nucleus) and optionally a second consonant, called the final (coda). The final (coda) can be optinally attached at the bottom of the character. Thus, if we want to train consonants and vowels separately and combine them for inference, we need 1) object detection and segmentation model to detect the consonants and vowels and 2) how to combine the detected letters in one character. Combining order matters in Korean!

8 Conclusion

In this work we evaluated Hangul recognition model for large number of classes (2,350). Even though we only tuned fc-layers of VGG-19 model, we could achieve fairly good performance for the classification accuracy. We also showed that the accuracy decreased as the number of classes increased. Hence fine-tuning method is right approach if size of datasets are reasonable. However, training a deeper network is encouraged for a large number of classes to capture more details.

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Bayesian network modeling of hangul characters for on-line handwriting recognition. *Proceedings of International Conference on Document Analysis and Recognition (ICDAR 2003)*, pages 207–211, 2003.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Sung-Jung Cho and Jin H. Kim. Neural network for hand-written character recognition using dynamic bar method. In *Proceedings of the Korea Information Science Autumn Conference*, volume 17, pages 251–254, 1990.

[5] Gyu-Ro Park, In-Jung Kim, and Cheng-Lin Liu. An evaluation of statistical methods in handwritten hangul recognition. *International Journal on Document Analysis and Recognition (IJDAR)*, 16(3):273–283, 2013.

[6] Ho Yon Kim and Jin H Kim. Handwritten korean character recognition based on hierarchical random graph modeling. In *Advances in Handwriting Recognition*, pages 347–356. World Scientific, 1999.

[7] In-Jung Kim and Xiaohui Xie. Handwritten hangul recognition using deep convolutional neural networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(1):1–13, 2015.

[8] Chung-Ang University Human-Computer Interaction Lab. Electronic and information research information center [https://www.eiric.or.kr/special/special.php], 2009.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[10] IBM. Handwritten korean character recognition with tensorflow and android. `https://github.com/IBM/tensorflow-hangul-recognition`, 2017.