Handwritten Hangul recognition using deep convolutional neural networks

In-Jung Kim[1] and Xiaohui Xie[2]

[1]School of CSEE, Handong Global University

791-708, Heunghae-eup, Bukgu, Pohang, Gyeongbuk, Republic of Korea

[2]Department of Computer Science, School of Information and Computer Science

University of California, 1 East Peltason Drive, Irvine, CA 92697, U.S.A.

Email: [1]ijkim@handong.edu, [2]xhx@ics.uci.edu

Tel: +82-54-260-1385, FAX: +82-54-260-1976

Abstract

In spite of the advances in recognition technology, handwritten Hangul recognition (HHR) remains largely unsolved due to the presence of many confusing characters and excessive cursiveness in Hangul handwritings. Even the best existing recognizers do not lead to satisfactory performance for practical applications, and have much lower performance than those developed for Chinese or alpha-numeric characters. To improve the performance of HHR, here we develop a new type of recognizers based on deep neural networks, which have recently shown excellent performance in many pattern recognition and machine learning problems, but have not been attempted for HHR. We build our Hangul recognizers based on deep convolutional neural networks, and propose several novel techniques to improve the performance and training speed of the networks. We systematically evaluated the performance of our recognizers on two public Hangul image databases, SERI95a and PE92. Using our framework, we were able to achieve a recognition rate of 95.96% on SERI95a, and 92.92% on PE92. Compared to the previous best records of 93.71% on SERI95a and 87.70% on PE92, our results provided improvements of 2.25% and 5.22%, respectively. These improvements lead to error reduction rates of 35.71% on SERI95a and 42.44% on PE92, relative to the previous lowest error rates. Such improvement fills a significant gap between practical requirement and the actual performance of Hangul recognizers.

Keywords: handwritten Hangul recognition, character recognition, deep convolutional neural network, deep learning, gradient-based learning

1. Introduction

Character recognition technology has been developed for decades. For alpha-numeric and Chinese characters, recognition technologies are mature enough to achieve high accuracy. However, in handwritten Hangul recognition (HHR), even the best recognizers cannot yield satisfactory performance for practical applications. The difficulty of handwritten Hangul recognition is mainly caused by a multitude of confusing characters and excessive cursiveness in Hangul handwritings. Figure 1 shows some examples of Hangul characters that are very similar and are often confused by recognizers. Hangul contains numerous such confusing characters. Moreover, shape variation in cursive handwritings makes it even harder to distinguish them. On the SERI95a and the PE92 databases, two most popular public Hangul image databases, the state-of-the-art performances in terms of recognition rate are merely 93.71% and 87.70%, respectively [1][2]. Such poor performance has discouraged the utilization of HHR in practical systems.

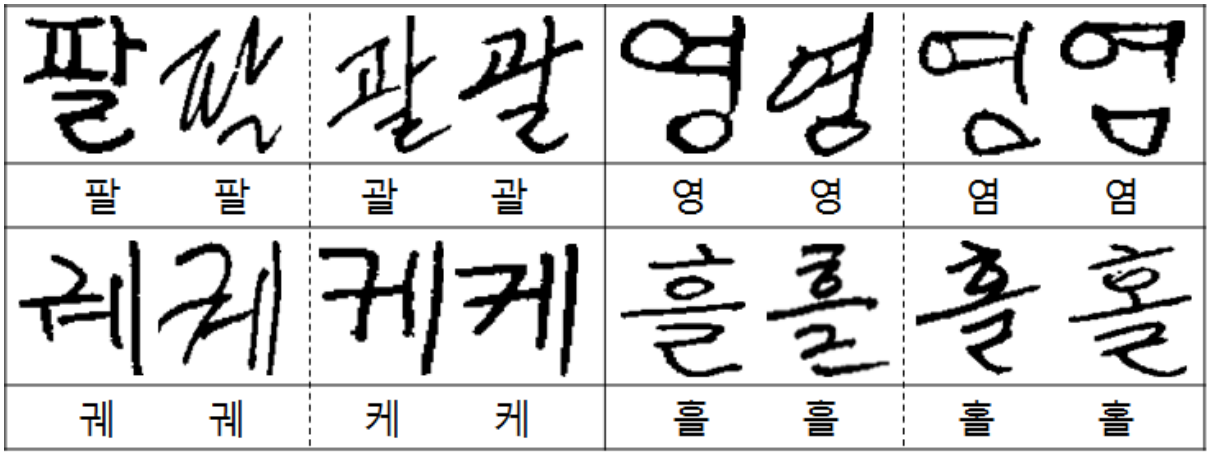


Figure 1. Examples of Hangul characters

On the other hand, in recent years, deep neural networks (DNN) have been highlighted in machine learning and pattern recognition fields. Composed of many layers, DNNs can model much more complicated functions than shallow neural networks [3]. The availability of large scale training data and advances in computing technologies have made the training of such deep networks possible, leading to a widespread adoption of DNNs in many problem domains. For

example, deep convolutional neural networks (DCNNs) have shown outstanding performance in many image recognition fields, beating benchmark performances by large margins [4][5][6][7]. However, although DNN has been employed for many pattern recognition systems, it has not been attempted for HHR. We reasoned that DNN could be especially beneficial for HHR for a number of reasons. First, DNN performs features learning and classification within a unified framework. Since features are automatically learned directly from the data themselves, it might be possible to extract subtle features to separate confusing characters in Hangul handwritings. Second, DNN is very good at extracting high-level features. In particular, the convolution and max-pooling layers used by DCNN have been shown to be very effective in handling shape variations, which will likely be key in handling the excessive cursiveness in Hangul writings. Therefore, DCNN seem to be well-posed to overcome the two main difficulties in HHR.

In this research, we build handwritten Hangul recognizers using DCNNs. Then, we improve the performance and the training speed of the recognizers by applying a few improvement techniques. Using the system we built, we are able to achieve a recognition rate of 95.26% on SERI95a and 92.92% on PE92. Compared to the previous best records of 93.71% on SERI95a and 87.70% on PE92, our results provided improvements of 2.25% and 5.22%, respectively. These improvements lead to error reduction rates of 35.71% on SERI95a and 42.44% on PE92, relative to the previous lowest error rates. Such improvement fills a significant portion of the large gap between practical requirement and the actual performance of Hangul recognizers.

The rest of this paper is organized as follows: In Section 2, we briefly review previous works on HHR and deep learning. In Sections 3 and 4, we describe the DCNN-based Hangul recognizer and the training algorithm. In Section 5, we propose several techniques to further improve performance and training speed. In Section 6, we present experimental results on two popular HHR datasets. Conclusions are provided in Section 7.

2. Related Works

2.1. Handwritten Hangul recognition

Character recognition methods can be generally grouped into two categories: structural and statistical. The structural method describes the input character as strokes or contour segments, and identifies the class by matching with the structural models of candidate classes. On the other hand, the statistical method represents the character image as a feature vector, and classifies the feature vector using statistical methodologies. Among them, statistical methods are more widely used in practical systems because they are easy to build and are effective in recognizing many character sets, including Chinese characters. However, unlike handwritten Chinese character recognition (HCCR), structural methods outperformed statistical methods in HHR for a long time. We believe the reason is that, in the early days, the statistical methods were insufficient to deal with the multitude of confusing characters and the excessive cursiveness in Hangul handwritings.

Kim and Kim proposed a structural method based on hierarchical random graph representation [8]. Given a character image, they extracted strokes and represented them onto an attributed graph, which is matched with character models using a bottom-up matching algorithm. Kang and Kim improved [8] by modeling between-stroke relationships [2]. They extended the hierarchical random graph in [8] by adding another type of nodes to represent relationships between strokes. Then, they matched those nodes with relationships among input strokes. Jang proposed a post-processing method for the structural recognizers in [8] and [2] to improve discrimination ability [9]. The post-processor consists of a set of pair-wise discriminators, each of which is specialized for a pair of graphemes with similar shapes. To build each pair-wise discriminator, they chose parts that separate the character pair, and then, applied statistical methods focusing on those parts. These systems were evaluated on two public handwritten Hangul image databases: SERI95a[1] and PE92

[1] SERI95a is also known as KU-1.

[10][11]. The best performances on SERI95a and PE92 achieved by the structural methods were 93.4% reported in [9] and 87.7% reported in [2], respectively.

In the early days of HHR, researchers attempted to use statistical methods to recognize handwritten Hangul [12][13][14]. However, the performance of those methods was much poorer than that of the structural methods, or it is hard to compare their performance to that of other methods because it was measured on small-size private datasets. As a result, statistical methods were not frequently used in HHR.

Meanwhile, statistical methods have become the mainstream approach in HCCR and have been improved significantly. Recently, Park et al. applied state-of-the-art statistical methods to HHR and evaluated their performance [1]. Combining non-linear shape normalization, the gradient feature extraction, and the modified quadratic discriminant function (MQDF) classifier, they achieved much better results than the early statistical recognizers. Their best performances were 93.71% on SERI95a and 85.99% on PE92, which are comparable to the performances of the structural recognizers. Specially, the recognition rate 93.71% on SERI95a is even higher than the best result of the structural method, 93.4%. Moreover, it is likely that the performance of statistical methods can be further improved when more training data are available [15]. However, at present neither the structural method nor the statistical method can provide a performance level high enough for practical applications. Consequently, handwritten Hangul recognition remains an unsolved problem.

2.2. Deep neural networks

For the past few years, DNNs have produced outstanding results in machine learning and pattern recognition fields. Composed of many layers, DNNs are much more efficient at representing highly varying nonlinear functions than shallow neural networks [3]. An additional reason for the good performance of DNNs is that DNNs enable integrated training of feature extractors and classifiers. Unlike conventional classifiers, most DNNs accept raw images as input,

and do not require separate feature extraction or preprocessing, except for size normalization. The low- and middle-level DNN layers extract and abstract the feature from the input image, while high-level layers perform classification. In this sense, a DNN can be viewed as a unified framework that integrates all modules within a single network that can be systematically optimized with respect to a single objective function. Such integrated training can often lead to better performance than those based on independent training of each module.

Despite their appealing properties in extracting and representing features, training DNNs is, however, computationally rather challenging. Back-propagation is the dominant algorithm used in training neural networks, where the error signals in the output layer of the network are propagated backward layer-by-layer from the output to the input layers to guide the update of connection weights. The back-propagation algorithm performs poorly when the number of hidden layers is large due to the so called "diminishing gradient problem" - as the error signals propagate backwards, they become smaller and smaller, and eventually become too small to guide the update of weights in the first a few layers. The diminishing gradient problem is a major obstacle in training DNNs.

However, in 2006, Hinton, et al. proposed a greedy layer-wise training algorithm to train the deep belief network (DBN) [16]. They first pre-trained the weights through an unsupervised training algorithm starting from the bottommost layer. Then, they fine-tuned the weights to minimize the classification error using a supervised training algorithm [17]. Their work made a breakthrough that vitalized deep learning research. Further, the idea of the unsupervised pre-training was applied to other neural networks such as the stacked auto-encoder [18].

Exceptionally, DCNNs can be trained with gradient-based learning algorithm without pre-training. The network structure was proposed by Fukushima in 1980 [19]. However, it has not been widely used because the training algorithm was not easy to use. In 1990s, LeCun et al. applied a gradient-based learning algorithm to DCNN and obtained successful results [20]. After that, researchers further improved DCNN and reported good results in image recognition [21].

Recently, Cireşan et al. applied multi-column DCNNs to recognize digits, alpha-numerals, Chinese characters, traffic signs, and object images [5][6]. They reported excellent results and surpassed conventional best records on many public databases, including MNIST digit image database, NIST SD19 alphanumeric character image database, and CASIA Chinese character image database.

In addition to the common advantages of deep neural networks, DCNN has some extra nice properties: It was designed to imitate human visual processing, and it has highly optimized structures to process 2D images. Further, DCNN can effectively learn the extraction and abstraction of 2D features. Particularly, the max-pooling layer of DCNN is very effective in absorbing shape variations. Moreover, composed of sparse connection with tied weights, DCNN has significantly fewer parameters than a fully connected network of similar size. Most of all, DCNN is trainable with the gradient-based learning algorithm, and suffers less from the diminishing gradient problem. Given that the gradient-based algorithm trains the whole network to minimize an error criterion directly, DCNN can produce highly optimized weights.

However, before now, DCNN has not been applied for recognizing handwritten Hangul characters. Several difficulties discourage the swift application of DCNNs in practical situations. Because of its complexity, implementing and debugging DCNN is time-consuming and difficult. Moreover, training a large DCNN requires heavy computation. For example, Cireşan et al. estimated that training a DCNN to recognize 3,755 Chinese characters on a single CPU would take more than one year [5]. Fortunately, the problem of the heavy computation can be partially alleviated by training with the GPU-based massive parallel processing [5][21].

3. The DCNN-based Hangul Recognizer

3.1. Overall structure

Figure 2 shows the overall structure of the DCNN. Each layer receives the output of the previous layer as its input, and passes the output to the next layer. We built the DCNN by combining three types of layers: convolution, max-pooling, and classification. The low- and

middle- level layers are composed of convolution and max-pooling layers alternately. The odd numbered layers, including the bottom layer, are composed of convolution layers, and the even numbered layers are composed of max-pooling layers. The nodes on the convolution layers and max-pooling layers are grouped into 2D planes, also called feature maps. Each plane is connected to one or more planes of the previous layer. Each node on a plane is connected to a small region of each connected plan in the previous layer. The node of the convolution layer extracts features from the input image (or input 2D feature maps) through the convolution operation on the input nodes, whereas the node of the max-pooling layer abstracts the features by propagating the maximum value among the input nodes.

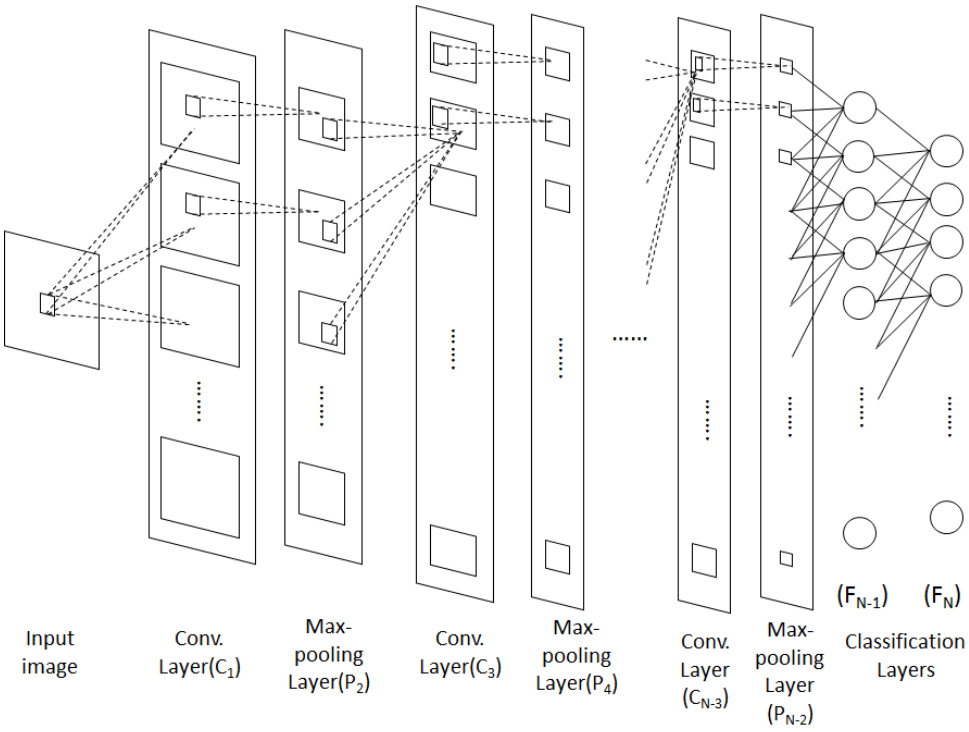


Figure 2. The overall architecture of the DCNN used by us, which includes an input layer, multiple alternating convolution and max-pooling layers, and two fully connected classification layers. N denotes the total number of layers in the network.

As the features are propagated to higher level layers, they are abstracted and combined to produce higher-level features. Meanwhile, the size of the feature map is reduced. In other words,

the higher the level, the smaller the size of the feature map. When the resolution of the feature map becomes 1x1 at a high-level layer, the features are passed onto the classification layers. The classification layers are placed at the top of the DCNN. They decide the classification result by analyzing the features extracted and abstracted by the preceding layers. For the classification layer, we applied the fully connected network because it is popular and has provided good performances in many recent works [5][21]. Each node of the top layer computes the score of a class. When the propagation finishes, the recognizer outputs the class with the highest score as the classification result.

3.2. Convolution layers

The convolution layer extracts features through the convolution operation on the input image or the feature maps of the previous layer. Each output plane is connected to one or more input planes. Each output node is connected to the input nodes in a small window. The horizontal and vertical distance between two adjacent windows is called stride. Denoting the stride by S, a node at (i, j) is connected to the input nodes in an MxM window whose upper left corner is at (iS, jS). Each node has a set of weights to connect itself to the input nodes. All nodes on a plane share the same weights.

Let $X^n_{(p,i,j)}$ denote the activation of a node at coordinate (i,j) on the p^{th} plane of the n^{th} layer and C^n_p denote the set of input planes connected to plane p at layer n. As all nodes on a plane share the same set of weights, the weight of the connection from $X^{n-1}_{(q,iS_n+u,jS_n+v)}$ to $X^n_{(p,i,j)}$ is simply denoted by $w^n_{(q,p,u,v)}$, where $0 \le u, v \le M_n - 1$, where M_n is the width and height of the convolution mask that connects layer *n-1* and layer *n*. The output of each node is computed as in equation (1), where θ^n_p is a bias, and f is an activation function.

$$X^n_{(p,i,j)} = f\left(\sum_{q \in C^n_p} \sum_{0 \le u,v \le M_n - 1} w^n_{(q,p,u,v)} X^{n-1}_{(q,iS_n+u,jS_n+v)} + \theta^n_p\right) \quad (1)$$

The first term in the argument of the activation function is a convolution operation with a mask composed of the shared weights. From this point of view, the nodes on a plane compute the same feature extracted from different locations. When the stride is set to one, the convolution layer extracts features from all possible coordinates. In this case, the convolution layer does not miss important features even if the positions of the features are shifted.

It is worth to note that equation (1) convolutes multiple input feature maps, thereby enabling the convolution layers to extract higher-level features from multiple lower-level features. The size of the output feature map after convolution is derived based on the size of the input feature map as shown in equation (2):

$$width^n = \lfloor (width^{n-1} - M_n + 1)/S_n \rfloor$$
$$height^n = \lfloor (height^{n-1} - M_n + 1)/S_n \rfloor \quad (2)$$

3.3. Max-pooling layers

The max-pooling layer abstracts the feature by pooling the input features. The output feature maps have one-to-one correspondence with the input feature maps. A node at (i, j) is connected to the input nodes in an MxM window whose upper left corner is at (iS, jS). Each node selects the maximum value among the input nodes as indicated by equation (3). Note that equation (3) does not require any weight.

$$X^n_{(p,i,j)} = f\left(\max_{0 \le u,v \le M_n - 1} X^{n-1}_{(p,iS_n+u,jS_n+v)}\right) \quad (3)$$

The average-pooling, frequently used for down-sampling, is an alternative to the max-pooling. However, a previous study reported that max-pooling produces better results than average-pooling [22]. Similar to the case of the convolution layer, the resolution of the output feature map is decided by equation (2). The stride of the max-pooling layer is often set to two. In this case,

the max-pooling layer reduces the size of the feature map approximately by a quarter. The max-pooling layer plays an important role: It absorbs shape variation or distortion. In handwritten characters, the positions of salient features often shift. The max-pooling node propagates only the maximum value in a window, ignoring the offset. Therefore, the max-pooling node catches the feature but ignores small displacements within the window. Given that a DCNN has a collection of max-pooling layers, each of which absorbs small positional shifts, the DCNN does not require a separate shape normalization step to regulate shape variation. Moreover, the max-pooling layers in a DCNN absorb shape variation in phases, which is desirable to minimize information loss.

3.4. Classification layers

Classification layers are placed at the top of the DCNN. They compute the score of each class from the features extracted and abstracted by the preceding layers. The size of the feature map is reduced to 1x1 at the last feature extraction or abstraction layer. Then, the feature maps are treated as scalar values and passed to the first fully connected layer. We used a fully connected feedforward network for the classification. The output of each node is computed as shown by equation (4), where the 2D coordinate (i,j) on each feature map is omitted because each feature map is composed of only one node in the final classification layers.

$$X_p^n = f\left(\sum_q w_{(q,p)}^n X_q^{n-1} + \theta_p^n\right) \tag{4}$$

3.5. Activation functions

Various types of activation functions are used in neural networks. In this research, we implemented sigmoid[23], hyperbolic tangent[24], softmax[25], rectified linear[26], and identity functions. In our preliminary experiments, the combination listed in Table 1 produced best results. Therefore we used this particular combination in our experiments.

Layer types	Activation functions
Convolution layers	identity
Max-pooling layers	rectified linear
Classification layers (hidden layer)	rectified linear
Classification layers (output layer)	tanh (for MSE criterion) or softmax (for cross entropy criterion)

Table 1. Activation functions for each layer type

4. Training DCNN

4.1. Gradient-based learning

The gradient-based learning algorithm in [20] is a generalization of the back-propagation algorithm, which iterates to adjust the weights to minimize an error function E. Starting from an initial weight vector W, it updates the weights as indicated by equation (5) at each iteration, where η is a learning rate.

$$W \leftarrow W - \eta \frac{\partial E}{\partial W} \tag{5}$$

Denoting the output and the weight vectors of the n^{th} layer as X^n and W^n, respectively, and applying the chain rule, the gradient at the n^{th} layer $\frac{\partial E}{\partial W^n}$ is expanded as demonstrated by equation (6).

$$\frac{\partial E}{\partial W^n} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial W^n} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial NET^n}\frac{\partial NET^n}{\partial W^n} \tag{6}$$

It is noteworthy that the product of the first two factors $\frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial NET^n}$ makes the error signal $\frac{\partial E}{\partial NET^n}$ used in the conventional back-propagation algorithm. In equation (6), $\frac{\partial X^n}{\partial NET^n}$ is the derivative of the activation function, and $\frac{\partial NET^n}{\partial W^n}$ is obtained from the input vector as listed in Table 2.

Layer types	Derivatives ($\frac{\partial NET^n}{\partial W^n}$)
Convolution	$\frac{\partial NET^n}{\partial w^n_{(q,p,u,v)}} = \sum_{i,j} X^{n-1}_{(q,iS_n+u,jS_n+v)}$
Max-pooling	N/A
Fully connected	$\frac{\partial net^n_p}{\partial w^n_{(q,p)}} = X^{n-1}_q$

Table 2. Derivatives of NET^N w.r.t. weights

The factor $\frac{\partial E}{\partial X^n}$ at the top layer is computed through the derivative of the error with respect to the output, as presented in the next section. However, those of other layers should be back-propagated from the upper layer. Therefore, each layer should compute $\frac{\partial E}{\partial X^{n-1}}$ as described equation (7) to provide to the lower layer.

$$\frac{\partial E}{\partial X^{n-1}} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial X^{n-1}} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial NET^n}\frac{\partial NET^n}{\partial X^{n-1}} \tag{7}$$

Note that X^{n-1} is the output of the (n-1)th layer as well as the input of the n^{th} layer. In equation (7), $\frac{\partial NET^n}{\partial X^{n-1}}$ is derived from the structure of the layer and the weight vector W^n, as listed in Table 3.

Layer types	Derivatives ($\frac{\partial NET^n}{\partial X^{n-1}}$)
Convolution	$\frac{\partial net^n_{(p,i,j)}}{\partial X^{n-1}_{(q,iS_n+u,jS_n+v)}} = w^n_{(q,p,u,v)}$, for all valid indices (i, j)s on output plane
Max-pooling	$\frac{\partial net^n_{(p,i,j)}}{\partial X^{n-1}_{(p,iS_n+u,jS_n+v)}} = 1$, if $(u,v) = \text{argmax}_{0\le\tilde{u},\tilde{v}\le M_n-1} X^{n-1}_{(p,iS_n+\tilde{u},jS_n+\tilde{u})}$ for the output node (i, j) $\frac{\partial net^n_{(p,i,j)}}{\partial X^{n-1}_{(p,iS_n+u,jS_n+v)}} = 0$, otherwise
Fully connected	$\frac{\partial net^n_p}{\partial X^{n-1}_q} = w^n_{(q,p)}$

Table 3. Derivatives of NET^N w.r.t. the input vector

As is the case with the conventional back-propagation algorithm, the gradient-based learning algorithm trains from the top layer to the bottom layer. At each layer, it computes equation (6) to update the weights as equation (5); then, it computes $\frac{\partial E}{\partial X^{n-1}}$ using equation (7) to back-propagate to the lower layer.

The gradient-based learning algorithm is applicable to a neural network that is composed of any types of layers for which $\frac{\partial E}{\partial W^n}$ and $\frac{\partial E}{\partial X^{n-1}}$ can be computed as given in equations (6) and (7), regardless of whether the layers are homogeneous or heterogeneous. The entire network function is not differentiable because of the max-pooling layer. However, the network function is still piece-wise differentiable, which is sufficient to apply the gradient-based learning.

There are several modes to train a neural network based on equation (5). The online mode training updates the weights with the gradient computed from each training sample. In contrast, the batch mode training first accumulates the gradients from all training samples, and then updates the weights with the accumulated gradient. The former is faster than the latter, but is less stable. On the other hand, the latter requires too much time to train a large DCNN with a large set of training samples. An intermediate, the mini-batch training, has a good balance between speed and stability. It partitions the training samples into groups, and updates the weights with the accumulated gradient obtained from each group.

4.2. Error criteria

Different error criteria lead to different objective functions for guiding the training process. Among them, the mean square error (MSE) is a popular error criterion. With a desired output D = (d_1, d_2, ... , d_C) for the training sample, MSE is defined as in equation (8), where X_c^N is the output of the top level layer for the c^{th} class and C is the number of classes.

$$E_{MSE} = \frac{1}{2}\frac{\sum_c (X_c^N - d_c)^2}{C} \tag{8}$$

The desired output is represented as follows: If c is the true class, d_c is one, otherwise, d_c is zero, for the unipolar activation function, or -1, for the bipolar activation function. The gradient of MSE with respect to the output is derived as in equation (9).

$$\frac{\partial E_{MSE}}{\partial X_c^N} = \frac{(X_c^N - d_c)}{C} \tag{9}$$

An alternative to MSE is the cross entropy (CE) error function. When used in conjunction with the softmax activation function, the CE has the form shown in equation (10).

$$E_{CE} = -\sum_c d_c \log(X_c^N) \tag{10}$$

While MSE minimizes the absolute error at each output node, CE maximizes the relative size of the true class output with respect to the outputs of other class nodes. CE is usually combined with the softmax activation function. Given that the softmax is a unipolar function, the desired output D consists of a single one for the true class, and zeroes for all other classes. The gradient of CE with respect to the output is computed as shown in equation (11).

$$\frac{\partial E_{CE}}{\partial X_c^N} = -\sum_c d_c \frac{1}{X_c^N} \tag{11}$$

4.3. Weight normalization

In order to avoid overfitting and to improve the generalization ability, we normalized the weights after each update. Weight normalization converts the incoming weights of each node into a unit vector [28]. Weight normalization of the convolution layer and the classification layer are as shown by equations (12) and (13), respectively. Given that the max-pooling layer does not have any weight, it does not require weight normalization.

$$w^n_{(q,p,u,v)} \leftarrow \frac{w^n_{(q,p,u,v)}}{\sqrt{\sum_{q,u,v} {w^n_{(q,p,u,v)}}^2}} \tag{12}$$

$$w^n_{(q,p)} \leftarrow \frac{w^n_{(q,p)}}{\sqrt{\sum_{q} {w^n_{(q,p)}}^2}} \tag{13}$$

There is an additional reason to normalize the weights. Unlike the sigmoid and the hyperbolic tangent functions, the outputs of the rectified linear and the identity activation functions are not bounded. Given that the network outputs affect the weight update, extreme output values can result in extreme weights. For this reason, the training of a DCNN with the rectified linear or the identity activation can be unstable. Weight normalization keeps the weights from diverging to extreme values.

5. Further Improvement Techniques

In order to further improve the performance and the training speed of DCNN, we applied a few additional techniques. Two of them are proposed in this research for the first time, whereas other two have been introduced in the literature.

5.1. Modified MSE criteria

A neural network-based recognizer with a large number of output classes is not easy to train with MSE. We attempted to train the DCNN recognizer with 520 output nodes, but our attempt was unsuccessful. Table 4 shows the improvement in MSE and recognition rate during the first 12 training epochs. Although we trained in the mini-batch mode, which is much faster than the batch mode, the decrease of MSE as well as the growth of the recognition rate were extremely slow. After 12 epochs, the recognition rate was 0.22%, which is only slightly better than random classification rate 1/520 = 0.19%.

The reason for the slow improvement can be found from the definition of MSE represented in equation (8). The desired output of the top-level layer is composed of many -1s but only a single one. The nodes of the hidden layers receive the signals back-propagated from all output nodes. The true class node sends a positive signal to cause the hidden nodes to encourage the activation of itself. However, the other C-1 output nodes send negative signals to cause the hidden nodes to discourage the activations of the other output nodes. The positive signal from the only true class node is not sufficiently strong to guide the training compared to the negative signals from the other 519 nodes. This problem is especially serious at the beginning of the training when the weights are not mature enough to compensate the unbalance in the strength of signals.

Epochs	MSE (eq.(8))	Recognition rate
1	0.004030	0.16%
2	0.002455	0.16%
3	0.002107	0.17%
4	0.002007	0.18%
5	0.001975	0.20%
6	0.001962	0.21%
7	0.001955	0.21%
8	0.001950	0.22%
9	0.001946	0.21%
10	0.001943	0.22%
11	0.001941	0.22%
12	0.001939	0.22%

Table 4. Result of Training DCNN-based Hangul recognizer using MSE criterion

To overcome this problem, we slightly modified the MSE criterion as shown by equation (14).

$$E_{MSE} = \frac{1}{2}\frac{\sum_c a_c (X_c^N - d_c)^2}{C} \quad \text{, where } \begin{cases} a_c = \alpha, & \text{if } c \text{ is the true class} \\ a_c = 1, & \text{otherwise} \end{cases} \tag{14}$$

Equation (14) is a generalization of equation (8) that assigns coefficient a_c to each class. α is an amplifying factor multiplied to the signal from the true class node. We can compensate the unbalance between the positive and the negative signals by setting α greater than one. Figure 3 shows the growth of the recognition rates when the DCNN was trained with various amplifying factors. The horizontal axis represents training epochs and the vertical axis represents the recognition rate on the training samples. With α = 1, which makes equation (14) equivalent to equation (8), the increase of the recognition rate for 20 epochs was almost negligible. Training with large amplifying factors increased the recognition rate much faster. Large amplifying factors were especially helpful in the early stages of the training.

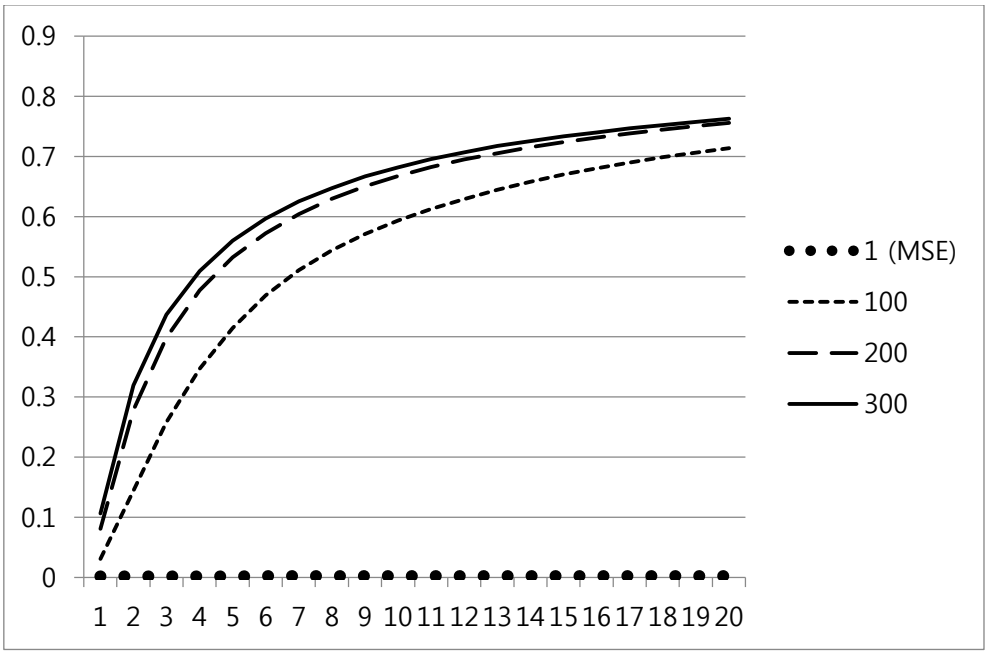


Figure 3. Training DCNN on SERI95a by modified MSE criterion
(The vertical axis represents recognition rate on training samples.)

However, the modified MSE with a large amplifying factor changes the objective function presented in equation (8). It can guide the training inappropriately. Fortunately, the signal unbalance problem becomes less serious as the weights mature. Therefore, we assigned a large number to α that could sufficiently compensate the signal unbalance at the beginning of the

training, and then, decreased it gradually as the training proceeds. When the training ends, α was reduced to one, which makes equation (14) no different from equation (8).

5.2. Initializing convolution masks by edge operators

In a deep neural network, the bottom layer is the most difficult to train with the top-down gradient-based algorithm because of the diminishing gradient problem described in section 2.2. Although the gradient-based learning algorithm on DCNN is less susceptible to the diminishing gradient problem than other DNNs, training bottom layer from random weights is not always the best way. As explained in section 3, the bottom level convolution layer extracts features from the input image. The gradient-based algorithm can train good feature extractors even from random initial weights. However, starting with good initial masks can help guide better feature extraction.

In the classical statistical recognition, researchers have achieved good performances by combining the contour directional feature extraction and QDF-based classification algorithms [1]. The set of eight directional gradient features is known as one of the best contour directional feature sets, and can be extracted by edge operators [29]. Inspired by the gradient feature extraction algorithm, we initialized the first eight convolution masks of the bottom layer with the 8-directional edge operators shown in Figure 4. As shown in the section 6, these initial masks were effective in improving the overall performance.

Figure 4. Edge operators to initialize convolution masks of the bottom layer

5.3. Elastic distortion

Many previous works reported that expanding training data set with artificially synthesized samples improved the performance [5][6][21]. The elastic distortion is an effective way to produce

artificial samples from the training samples [21][30]. The distortion algorithm distorts the image by shifting each pixel's coordinate according to a distortion map. We applied the algorithm in [21] to build the distortion map. First, it generates pairs of random numbers between -1 and 1 for the horizontal and vertical displacements of all pixels. Then, it convolves the displacement field with a Gaussian of standard deviation σ to avoid drastic deformation, and normalize the displacement field to a norm of one. Finally, it multiplies the displacement field by a scaling factor s. In the experiments, we set σ by four and s by one. For details, see [21].

In the training, we generated a distortion map whenever a new mini-batch group began and applied it to all the samples in the group. In preliminary experiments, this method showed better results than generating a new distortion map for each sample. We believe the reason is that generating one distortion map for each sample causes the DCNN to confuse the shape variation with salient information for the recognition.

5.4. GPU-based parallel processing

Accelerating training speed by GPU-based parallel processing is essential to train a DCNN-based recognizer for a large character set [5]. The full set of Hangul contains 11,172 characters and 2,350 characters among them are used daily. The PE92 database contains 2,350 classes and the SERI95a database contains 520 classes. It takes days to train the DCNN-based Hangul recognizer for a single epoch on a CPU. Training for hundreds epochs on a CPU would take years.

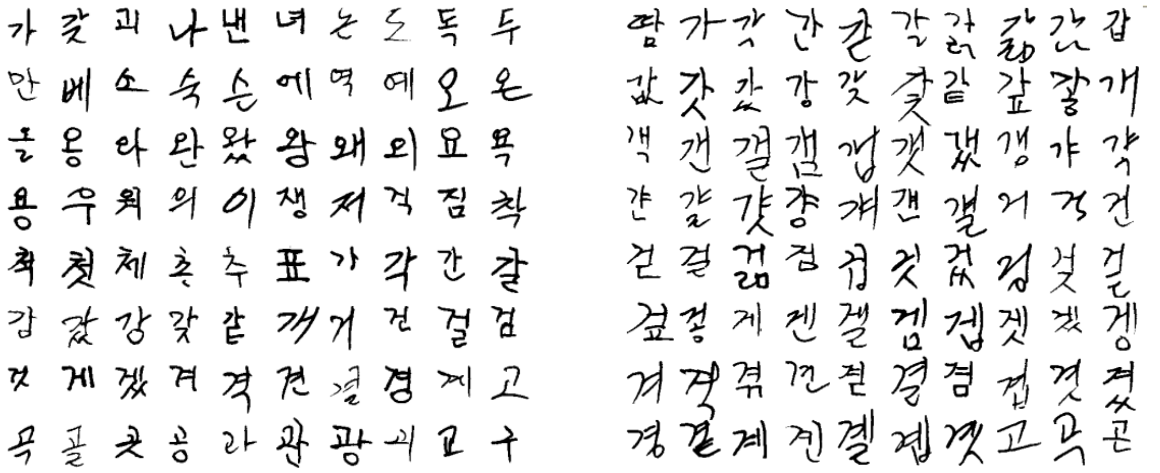
Recent high-end GPUs contain thousands of computing units. Composed of many nodes, neural networks are appropriate to exploit the benefits of massive parallel processing. We applied NVDIA CUDA to run the training algorithm on GPU. The improvement of the training speed heavily depends on the parallelism of the network structure. On a narrow network composed of a small number of hidden nodes, the GPU-based parallel processing demonstrates little improvement. However, on a broad neural network containing a large number of hidden nodes, it makes the training much faster. In Hangul recognition, training an epoch takes about 1.25 hours

on GTX Titan, which is about 20 times faster than the serial algorithm written in highly optimized C++ codes. With GPU-based parallel processing, training a Hangul recognizer for 500 epochs consumes 625 hours, which is about 26 days.

6. Experiments

6.1. Experimental environment

We evaluated the DCNN-based recognizers on the PE92 and the SERI95a databases. PE92 contains 2,350 character classes each of which has about 100 samples. SERI95a has 520 most frequently used character classes and each class contains about 1000 samples. Some examples are presented in Figure 5. For fair evaluation, we chose the training and the test sets in the same way as [1]. We used every 10^{th} sample of each class for the test and all other samples for the training. Thus, the training and the test sets contain 90% and 10% of total samples, respectively.



(a) SERI95a (b) PE92

Figure 5. Example images in SERI95a and PE92 databases

We described two training criteria and various improvement methods in the previous sections. These training criteria and improvement methods can be combined in several ways. Considering the amount of time required to train a Hangul recognizer, testing all possible cases on the Hangul

databases is significantly time-consuming even on a GPU. Therefore, we first tested all combinations on the MNIST handwritten digit database [31], which is much less time-consuming. Then, we tested only meaningful combinations on the two Hangul databases.

We experimented on six computers with CPUs that varied from Q6600 2.4 GHz to ZEON E3-1230V3 3.3 GHz. The GPUs are also varied from GTX 660Ti (1,344 CUDA cores, 2GB RAM) to GTX Titan (2,688 CUDA cores, 6GB RAM). For the GPU-based implementation, we used CUDA SDK v.5.5. In all experiments, we trained in the mini-batch mode.

6.2. Numeral digit recognition (MNIST)

The input of the digit recognizer is a 32x32 image that contains a 28x28 digit image at the center and four padding rows and columns at the boundary. The resolutions of the feature maps were decided by equation (2). The digit recognizer is composed of seven layers. The feature maps of each convolution layer are fully connected to all feature maps of the previous layer. The detail of the network structures is presented in Table 5. The DCNN has 299,882 parameters, totally.

layer	type	# of feature maps	feature map size	window size	stride	# of parameters
C_1	convolution	32	28x28	5x5	1	832
P_2	max-pooling	32	14x14	2x2	2	0
C_3	Convolution	32	10x10	5x5	1	25,632
P_4	max-pooling	32	5x5	2x2	2	0
C_5	Convolution	256	1x1	5x5	1	205,056
F_6	fully connected	256	1x1	N/A	N/A	65,792
F_7	fully connected	10	1x1	N/A	N/A	2,570

Table 5. Structure of digit recognizer (MNIST)

We tested the two error criteria as well as the two improvement techniques described in sections 4.2, 5.2, and 5.3. In this experiment, we evaluated all possible eight cases. For each case, we trained for 1,000 epochs, which took several days on a GPU. The experiment results are

presented in Table 6. Regarding the error criteria, MSE was slightly better than CE when we trained without distortion. However, using elastic distortion, CE showed better results. Overall, CE was slightly better than MSE in the best performance values. The elastic distortion significantly reduced error rates in all cases. Setting initial convolution masks by the edge operators further improved the recognition performance. The best performance we achieved was 99.67%, obtained by training with the CE criterion and elastic distortion starting from the convolution masks initialized by the edge operators.

	MSE		**CE**	
	recog. rate	**error rate**	**recog. rate**	**error rate**
baseline	99.29%	0.71%	99.22%	0.78%
edge operators	99.31%	0.69%	99.28%	0.72%
elastic distortion	99.51%	0.49%	99.63%	0.37%
edge operators + elastic distortion	99.65%	0.35%	99.67%	0.33%

Table 6. Digit recognition results

After we obtained the results listed in Table 6, we continued to train the best combination. The recognition rate increased even after the 1,000th epoch. However, the increment was not significant. After training for 3,000 epochs, we achieved 99.71% of recognition rate (0.29% of error rate). The homepage of the MNIST database lists the best performances on their database achieved by various methods [31]. The lowest error rate in the list is 0.23%, not far from our result of 0.29%. Only two systems in the list reported better results than ours. The best two results were achieved by committees of many DCNNs, not by single classifiers.

6.3. Hangul recognition (SERI95a and PE92)

The input of the Hangul recognizers is a 64x64 image that consists of a 60x60 Hangul image and four padding rows/columns. The Hangul recognizers are composed of ten layers. Similar to the digit recognizer, the feature maps of each convolution layer are fully connected to all feature maps of the previous layer. The DCNNs for the two Hangul databases are different in the number

of nodes at the highest two layers. The details of the network structures are described in Table 7 and Table 8. The DCNNs have a total of 1,006,728, and 1,106,184 parameters, respectively.

layers	type	# of feature maps	feature map size	window size	stride	# of parameters
C_1	convolution	32	60x60	5x5	1	832
P_2	max-pooling	32	30x30	2x2	2	0
C_3	convolution	64	26x26	5x5	1	51,264
P_4	max-pooling	64	13x13	2x2	2	0
C_5	convolution	128	10x10	4x4	1	131,200
P_6	max-pooling	128	5x5	2x2	2	0
C_7	convolution	256	2x2	4x4	1	524,544
P_8	max-pooling	256	1x1	2x2	1	0
F_9	fully connected	384	1x1	N/A	N/A	98,688
F_{10}	fully connected	520	1x1	N/A	N/A	200,200

Table 7. Structure of Hangul recognizer (SERI95a)

layers	type	# of feature maps	feature map size	window size	stride	# of parameters
C_1	convolution	32	60x60	5x5	1	832
P_2	max-pooling	32	30x30	2x2	2	0
C_3	convolution	64	26x26	5x5	1	51,264
P_4	max-pooling	64	13x13	2x2	2	0
C_5	convolution	128	10x10	4x4	1	131,200
P_6	max-pooling	128	5x5	2x2	2	0
C_7	convolution	256	2x2	4x4	1	524,544
P_8	max-pooling	256	1x1	2x2	1	0
F_9	fully connected	512	1x1	N/A	N/A	131,584
F_{10}	fully connected	2,350	1x1	N/A	N/A	266,760

Table 8. Structure of Hangul recognizer (PE92)

Given that training Hangul recognizers requires a significant amount of time, and we know that the methods described in sections 5.2 and 5.3 are helpful in improving performance, we did not test all possible combinations of experiment options. Instead, we tested the improvement methods incrementally. For each case, we trained for 500 epochs. We applied the method

introduced in section 5.1 to train DCNNs with the MSE criterion. The amplifying factor α was set to 300 when the training started, and linearly decreased to one.

	SERI95a (520 classes)		PE92 (2,350 classes)	
	recog. rate	**error rate**	**recog. rate**	**error rate**
A. baseline (MSE)	88.96%	11.04%	74.93%	28.72%
B. baseline (CE)	95.55%	4.45%	91.44%	8.56%
C. edge operators (CE)	95.78%	4.22%	91.78%	8.22%
D. edge operators + elastic distortion (CE)	95.96%	4.04%	92.92%	7.08%

Table 9. Hangul recognition results

Table 9 presents the results. Unlike the digit recognition results, the results of the MSE criterion are significantly inferior when compared to those of the CE criterion. We believe the reason is that minimizing the absolute error of each output node is inefficient in training a recognizer for hundreds or thousands of classes. Similar to the previous experiment, initializing convolution masks by the edge operators and applying elastic distortion improved the performance. In order to know whether the improvements are statistically significant, we carried out McNemar's test and Chi-square test on the results on SERI95a database. In McNemar's test, the pvalues of the improvements by cross entropy criterion (A->B), edge operator (B->C), and elastic distortion (C->D) were 0.00E+00, 2.45E-03, and 2.42E-06, respectively. In Chi-square test, pvalues were 0.00E+00, 3.48E-02, and 3.48E-02. These pvalues show that the improvements are statistically significant. Particularly, the effect of the elastic distortion on the PE92 database was more remarkable than that on the SERI95a database. This is because PE92 contains more classes but less samples per class; therefore, the recognizer considerably suffers from a lack of training samples.

Researchers	Recognition methods	SERI95a	PE92
Kim&Kim2001 [8]	Structural matching	86.30%	82.20%
Kang&Kim2004 [2]	Structural matching	90.30%	87.70%
Jang&Kim2002 [9]	Structural matching + Post-processing	93.40%	N/A
Park, et. al. 2013 [1]	MQDF	93.71%	85.99%
Proposed	**DCNN**	**95.96%**	**92.92%**

Table 10. Recognition rates achieved on SERI95a and PE92 databases

Table 10 compares our results with previous best works on SERI95a and PE92. The elastic distortion was not used listed in the previous works in Table 10. As underlined in Table 10, the conventional best performances on SERI95a and PE92 databases were 93.71% and 87.70%, respectively. Surprisingly, most of the results in Table 9 are better than the conventional best performances. Specifically, our best performances are noticeably higher than the two conventional best results. Compared to the previous best records of 93.71% on SERI95a and 87.70% on PE92, our results provided improvements of 2.25% and 5.22%, respectively. These improvements lead to error reduction rates of 35.71% on SERI95a and 42.44% on PE92, relative to the previous lowest error rates.

7. Conclusion

In spite of the advances in recognition technology, handwritten Hangul recognition (HHR) has remained largely unsolved due to the presence of many confusing characters and excessive cursiveness in Hangul handwritings. On the other hand, the DCNN has provided outstanding performances in many recognition fields. However, before now, the DCNN has not been applied to recognize handwritten Hangul. In this research, we built handwritten Hangul recognizers using DCNNs and evaluated their performances on the SERI95a and the PE92 databases. Then, we improved the training speed and the recognition performance through GPU-based parallel processing and elastic distortion.

We also proposed two new improvement techniques. The modified MSE error criterion significantly improved the training efficiency of the Hangul recognizer by compensating the unbalance between positive and negative signals from the output nodes. Additionally, we achieved further improvement by initializing bottom level convolution masks by edge operators. Training convolution masks starting from good initial weights was helpful in obtaining good feature extractors.

In the experiments, we achieved recognition rates 95.96% on SERI95a and 92.92% on PE92, which are significantly higher than conventional best records. In handwritten digit recognition, and we achieved 99.71% recognition rate on the MNIST database.

References

[1] G.-R. Park, I.-J. Kim, C.-L. Liu, An evaluation of statistical methods in handwritten Hangul recognition, Int. J. Document Analysis and Recognition, vol. 16, no. 3, pp. 273-283, 2013.

[2] K.-W. Kang and J.H. Kim, Utilization of hierarchical, stochastic relationship modeling for Hangul character recognition, IEEE Trans. Pattern Analysis and Machine Intelligence, vol.26, no.9, pp.1185-1196, 2004.

[3] Y. Bengio, Learning deep architectures for AI, Foundations and Trends in Machine Learning, vol. 2, iss. 1, pp. 1-127, 2009.

[4] Cheng-Lin Liu, Fei Yin, Qiu-Feng Wang, Da-Han Wang, ICDAR 2011 Chinese Handwriting Recognition Competition, 2011. (http://www.nlpr.ia.ac.cn/events/HRcompetition/Report.html)

[5] D. C. Cireşan, U. Meier, J. Schmidhuber. Multi-column Deep Neural Networks for Image Classification. IEEE Conf. on Computer Vision and Pattern Recognition 2012.

[6] D. C. Cireşan and J. Schmidhuber. Multi-column Deep Neural Networks for Offline Handwritten Chinese Character Classification, IDSIA Technical Report No. IDSIA-05-13, 2013.

[7] ImageNet Large Scale Visual Recognition Challenge 2013 (ILSVRC2013) Result (http://www.image-net.org/challenges/LSVRC/2013/results.php)

[8] H. Y. Kim and J. H. Kim, Hierarchical random graph representation of handwritten characters and its application to Hangul recognition, Pattern Recognition, vol. 34, no. 2, pp.187-201,

2001.

[9] S. I. Jang, Post-processing of handwritten Hangul recognition using pair-wise grapheme discrimination, Master Thesis, KAIST, 2002.

[10] D. -I. Kim and S. -W. Lee, Automatic Evaluation of Handwriting Qualities of Handwritten Hangul Image Database, KU-1, Proc. 6th IWFHR, Taejon, Korea, pp. 455-464, Aug. 1998.

[11] D. H. Kim, Y. S. Hwang, S. T. Park, E. J. Kim, S. H. Paek, and S. Y. Bang, Handwritten Korean character image database PE92, Proc. 2nd ICDAR, pp. 470-473, Oct. 1993.

[12] H. J. Bae, J. M. Yun, and E. Y. Cha, Neural network for hand-written character recognition using dynamic bar method, Proc. Korea Information Science Autumn Conference, vol.17, no.2, pp.251-254, 1990.

[13] M. W. Kim, J. S. Jang, C. D. Lim, Y. S. Song, and J. H. Kim, Improvements to a hierarchical interaction neural network for context-dependent pattern recognition and its experimentation with handwritten Korean character recognition, Technical Report, Electronics and Telecommunication Research Institute, Taejon, Korea, 1992.

[14] S. H. Jeong, Handwritten Hangul recognition based on character cluster segmentation, Technical Memo, Electronics and Telecommunication Research Institute, Taejon, Korea, 2002.

[15] A. Torralba, R. Fergus, and W. Freeman, 80 million tiny images: a large dataset for non-parametric object and scene recognition, IEEE Trans. Pattern Analysis and Machine Intelligence vol. 30, no. 11, pp. 1958-1970, 2008..

[16] G. E. Hinton, S. Osindero, Y. Teh, A fast learning algorithm for deep belief nets. Neural Computation vol. 18, no. 7, pp. 1527-1554, 2006.

[17] G. E. Hinton, P. Dayan, B. J. Frey, and R. Neal, The wake-sleep algorithm for self-organizing neural networks. Science, 268, pp. 1158-1161, 1995.

[18] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08), (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 1096–1103, ACM, 2008.

[19] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, vol. 36, no. 4, pp. 193–202, 1980.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document

recognition, Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] D. Simard, P. Y. Steinkraus, and J. C. Platt, Best practices for convolutional neural networks, Proc. International Conference on Document Analysis and Recognition (ICDAR'03), p. 958, Washington, DC, USA: IEEE Computer Society, 2003.

[22] Y. Boureau, J. Ponce, and Y. LeCun, A theoretical analysis of feature pooling in vision algorithms, Proc. International Conference on Machine learning (ICML'10), 2010.

[23] http://en.wikipedia.org/wiki/Sigmoid_function

[24] C. Ozkan, and F. Erbek. A Comparison of activation functions for multispectral Landsat TM image classification. Photogrammetric engineering and remote sensing 69.11 (2003): 1225-1234.

[25] http://en.wikipedia.org/wiki/Softmax_activation_function

[26] V. Nair and G. Hinton, Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th International Conference on Machine Learning (ICML-10). 2010.

[27] X. Glorot, B. Antoine, and Y. Bengio, Deep Sparse Rectifier Networks. Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume. Vol. 15. 2011.

[28] G. Goodhill and H. Barrow, The role of weight normalization in competitive learning. Neural Computation vol. 6, no. 2, pp. 255-269, 1994.

[29] H. Liu and X. Ding, Handwritten character recognition using gradient feature and quadratic classifier with multiple discrimination schemes, Proceedings. 8th International Conference on Document Analysis and Recognition, IEEE, 2005.

[30] D. Cireşan, et al., Deep Big Multilayer Perceptrons for Digit Recognition, Neural Networks: Tricks of the Trade. Springer Berlin Heidelberg, pp. 581-598, 2012.

[31] http://yann.lecun.com/exdb/mnist