IJDAR (2015) 18:1–13
DOI 10.1007/s10032-014-0229-4

ORIGINAL PAPER

Handwritten Hangul recognition using deep convolutional neural networks

In-Jung Kim · Xiaohui Xie

Received: 10 February 2014 / Revised: 15 August 2014 / Accepted: 20 August 2014 / Published online: 9 September 2014

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 alphanumeric characters. To improve the performance of HHR, here we developed a new type of recognizers based on deep neural networks (DNNs). DNN has recently shown excellent performance in many pattern recognition and machine learning problems, but have not been attempted for HHR. We built our Hangul recognizers based on deep convolutional neural networks and proposed 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 achieved a recognition rate of 95.96 % on SERI95a and 92.92 % on PE92. Compared with the previous best records of 93.71 % on SERI95a and 87.70 % on PE92, our results yielded 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.

I.-J. Kim (✉)
School of CSEE, Handong Global University, 791-708, Heunghae-eup, Bukgu, Pohang, Gyeongbuk, Republic of Korea
e-mail: ijkim@handong.edu

X. Xie
Department of Computer Science, School of Information and Computer Science, University of California, 1 East Peltason Drive, Irvine, CA 92697, USA
e-mail: xhx@ics.uci.edu

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 alphanumeric and Chinese characters, recognition technologies are mature enough to achieve high accuracy. However, in handwritten Hangul recognition (HHR), none of existing recognizers are accurate enough 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 a lot of such confusing characters. Moreover, shape variation in cursive handwritings makes it even harder to distinguish them. On the SERI95a and PE92 databases, two most popular public Hangul image databases, the state-of-the-art performances in terms of recognition rates are merely 93.71 and 87.70 %, respectively [1,2]. Such poor performances have discouraged the utilization of HHR in practical systems.

On the other hand, in recent years, deep neural networks (DNNs) have been highlighted in machine learning and pattern recognition fields. Composed of many layers, DNNs can model much more complicated functions than shallow 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 performances in many image recognition fields, beating benchmark performances by large margins [4–7].

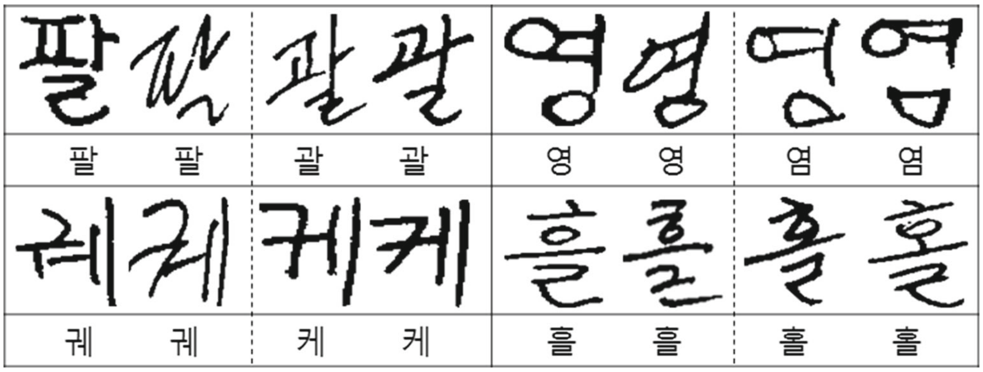

Fig. 1 Examples of Hangul characters

However, although DNN has been employed for many pattern recognition systems, it has not been attempted for HHR. We reason that DNN could be especially beneficial for HHR for a number of reasons. First, DNN integrates feature extraction and classification within a unified framework. Such an integrated structure can be advantageous in learning discriminative features necessary to distinguish confusing characters in Hangul handwritings, because features are systematically learned together with the classifier within the framework of error minimization, and such features are important in error minimization. Second, DNN is very good at extracting high-level features. In particular, the convolution and max-pooling layers used by DCNN are very effective in handling shape variations, which will likely be key in handling the excessive cursiveness in Hangul writings. Therefore, DCNN seems to be well posed to overcome the two main difficulties in HHR.

In this research, we built handwritten Hangul recognizers using DCNNs. Then, we improved the performance and the training speed of the recognizers by applying a few improvement techniques. Using the system we built, we achieved a recognition rate of 95.26 % on SERI95a and 92.92 % on PE92. Compared with the previous best records of 93.71 % on SERI95a and 87.70 % on PE92, our results yielded 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 Sect. 2, we briefly review previous works on HHR and deep learning. In Sects. 3 and 4, we describe the DCNN-based Hangul recognizer and the training algorithm. In Sect. 5, we propose several techniques to further improve performance and training speed. In Sect. 6, we present experimental results on two popular HHR data sets. Conclusions are provided in Sect. 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. In contrast, the statistical method represents the character image as a feature vector and classifies the feature vector using statistical methodologies. Between the two, the statistical method is more widely used in practice because it is easy to build and is 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 the statistical methods used in early days were not sophisticated and therefore 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 the 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 extracted 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 [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 [12–14]. However, the performances of those statistical methods were either much poorer than those of the structural methods, or could not be directly compared, because they were measured on small-size private data sets. As a result, statistical methods were not frequently used in HHR for a while.

On the other hand, statistical methods have become the mainstream approach in HCCR and have been improved significantly. Recently, Park et al. [1] applied state-of-the-art statistical methods to HHR and evaluated their performance. Combining nonlinear 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. Especially, the recognition rate 93.71 % on SERI95a is even higher than the best result of the structural method, 93.4 %. Moreover, it is possible that the performance of statistical methods can be further improved when more training data become 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, DNN has produced outstanding results in machine learning and pattern recognition fields. Composed of many layers, DNN is much more efficient at representing highly varying nonlinear functions than shallow neural networks [3]. In addition, 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 features from the input image, while high-level layers perform classification. As such, a DNN integrates feature extraction and classification within a single network, which can be systematically optimized with respect to a single objective function. Such integrated training can often lead to better performance than those based on the independent training of each module.

Despite their appealing properties in extracting and representing features, training DNNs are, however, computationally challenging. Back propagation is the dominant algorithm used in training neural networks. In the back-propagation training, the error signals in the output layer of the network are propagated backward layer by layer from the output layer to the input layer 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 backward, they become smaller and smaller and eventually become too small to guide the update of weights in the lowest few layers. The diminishing gradient problem is a major obstacle in training of DNNs.

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

Exceptionally, DCNN can be trained with gradient-based learning algorithm even 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. [20] applied a gradient-based learning algorithm to DCNN and obtained successful results. 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, alphanumerals, 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 therefore, it has a highly optimized structure 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

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

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 DCNN in practical situations. Because of its complexity, implementing and debugging DCNN is difficult and time-consuming. Moreover, training a large DCNN requires heavy computation. For example, Cireşan et al. [5] estimated that training a DCNN to recognize 3,755 Chinese characters on a single CPU would take more than 1 year. 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 convolution 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 on the connected input planes. 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.

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, we applied the fully connected network, because it is popular and has provided good performances in several recent works [5,21]. Each node of the top-level layer computes the score of a class. When the propagation finishes, the recognizer outputs the class with the highest score as the classification result.

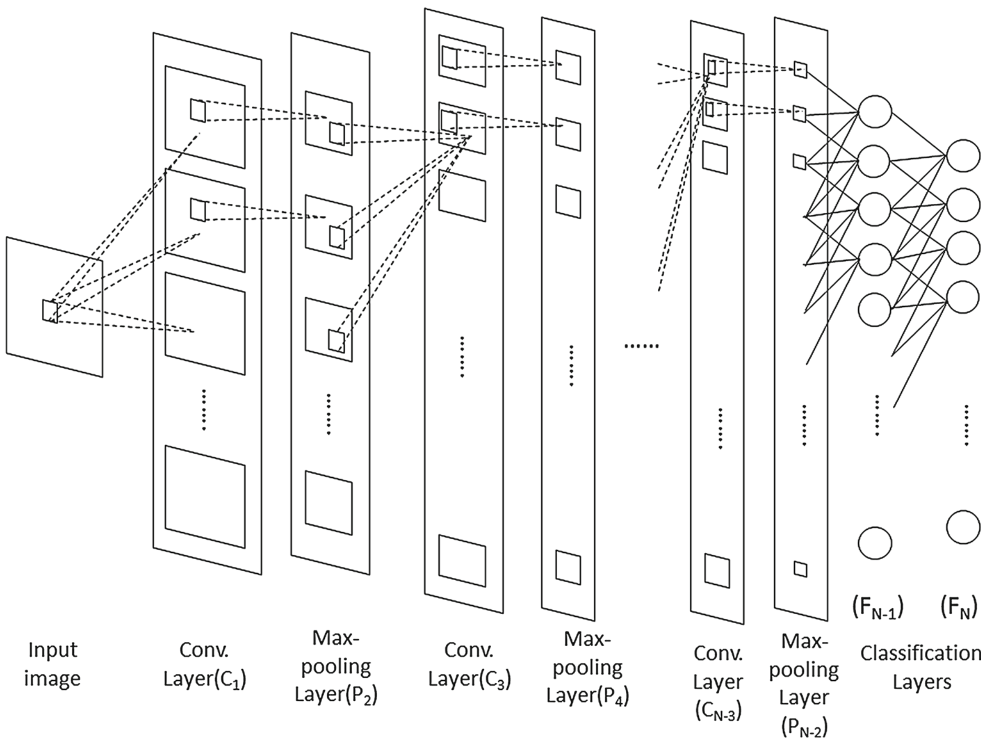

Fig. 2 The overall architecture of the DCNN used by us, which includes an input layer, multiple alternating convolution and max-pooling layers, and two fully connected classification layers. *N* denotes the total number of layers in the network

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 $M \times M$ 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 pth plane of the nth layer and C^n_p denote the set of input planes connected to plane p of 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 $0 \le u, v \le M_n - 1$, with M_n being the width and height of the convolution mask connecting layers $n-1$ and n. The output of each node is computed as in Eq. (1), where θ^n_p is a bias, and f is an activation function.

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

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 noting that Eq. (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 Eq. (2):

$$\begin{aligned} \text{width}^n &= \lfloor (\text{width}^{n-1} - M_n + 1)/S_n \rfloor \\ \text{height}^n &= \lfloor (\text{height}^{n-1} - M_n + 1)/S_n \rfloor \end{aligned} \tag{2}$$

3.3 Max-pooling layers

The max-pooling layer abstracts the input feature into a lower dimensional feature. 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 $M \times M$ window whose upper left corner is at (iS, jS). Each node selects the maximum value among the input nodes as indicated by Eq. (3). Note that Eq. (3) does not require any weight.

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

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 Eq. (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 1×1 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 the fully connected feed-forward network for the classification. The output of each node is computed as shown by Eq. (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.

$$X^n_p = f\left(\sum_q w^n_{(q,p)} X^{n-1}_q + \theta^n_p\right) \tag{4}$$

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 [26,27], 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.

Table 1 Activation functions for each layer type

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)

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 Eq. (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 nth layer as X^n and W^n, respectively, and applying the chain rule, the gradient at the nth layer $\frac{\partial E}{\partial W^n}$ is expanded as demonstrated by Eq. (6).

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

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

Layer types	Derivatives $\left(\frac{\partial NET^n}{\partial W^n}\right)$
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	$\frac{\partial net^n_p}{\partial w^n_{(q,p)}} = X^{n-1}_q$

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 Eq. (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.

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 their upper layers. 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 nth layer. In Eq. (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.

Similar to 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 Eq. (5); then, it computes $\frac{\partial E}{\partial X^{n-1}}$ using Eq. (7) to back-propagate to the lower layer.

The gradient-based learning algorithm is applicable to neural networks 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 Eqs. (6) and (7), regardless of whether the layers are homogeneous or heterogeneous. Although the entire network function is not differentiable because of the max-pooling layer, it is still piece-wise differentiable, and therefore, the gradient-based learning is still applicable.

There are several modes to train a neural network based on Eq. (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 computed from all training samples and then updates the weights with the accumulated gradient. The former decreases the error more quickly than the latter, but is less stable. On the other hand, the latter requires very long 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

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

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 $(u, v) = \underset{0\le u,v\le M_n-1}{\operatorname{argmax}} X^{n-1}_{(p,iS_n+u,jS_n+u)}$ 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	$\frac{\partial net^n_p}{\partial X^{n-1}_q} = w^n_{(q,p)}$	

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 choice. With a desired output $D = (d_1, d_2, \ldots, d_C)$ for the training sample, MSE is defined as in Eq. (8), where X_c^N is the output of the top-level layer for the cth class, and C is the number of classes.

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

The desired output is represented as follows: d_c is one, if c is the true class, and 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 Eq. (9).

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

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 Eq. (10).

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

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 Eq. (11).

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

4.3 Weight normalization

In order to avoid overfitting and to improve the generalization ability, we normalized the weights after each update. Weight normalization scales 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 Eqs. (12) and (13), respectively. Given that the max-pooling layer does not have any weight, it does not require weight normalization.

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

$$w_{(q,p)}^n \leftarrow \frac{w_{(q,p)}^n}{\sqrt{\sum_q {w_{(q,p)}^n}^2}} \tag{13}$$

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 function 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 was extremely slow.

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

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

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 Eq. (8). The desired output of the top-level layer is composed of many -1 s 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 with 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.

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

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

Equation (14) is a generalization of Eq. (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 $\alpha = 1$, which makes equation (14) equivalent to Eq. (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.

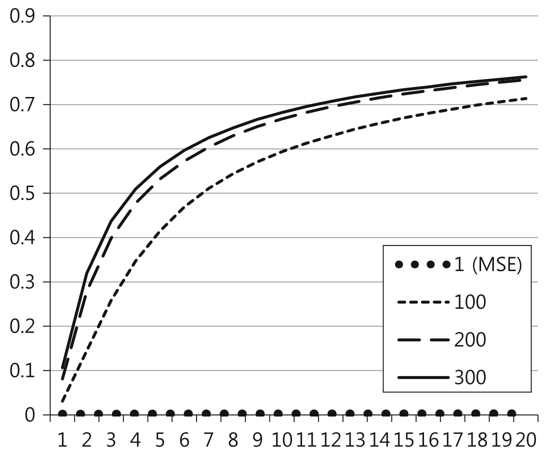

Fig. 3 Training DCNN on SERI95a by modified MSE criterion (the *vertical axis* represents recognition rate on training samples)

However, the modified MSE with a large amplifying factor changes the objective function presented in Eq. (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 Eq. (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 Sect. 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 Sect. 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 to find better feature extractors.

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 Fig. 4. As shown in the Sect. 6, these initial masks were effective in improving the overall performance.

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 normalizes the

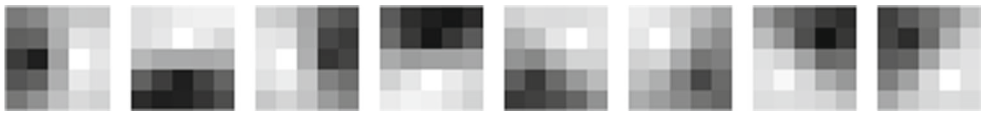

Fig. 4 Edge operators used to initialize convolution masks of the bottom layer

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 SDK 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 network containing a large number of hidden nodes, it accelerates training significantly. In Hangul recognition, training an epoch takes about 1.25 h on GTX Titan, which is about 20 times faster than the serial implementation written in highly optimized C++ codes. With GPU-based parallel processing, training a Hangul recognizer for 500 epochs consumes 625 h, 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 1,000 samples. Some examples are presented in Fig. 5. For fair evaluation, we chose the training and the test sets in the same way as [1]. We used every 10th 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.

We described two training criteria and several improvement techniques in the previous sections. These training criteria and improvement techniques can be combined in various 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 are 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 32×32 image that contains a 28×28 digit image at the center and four padding rows and columns at the boundary. The resolutions of the feature maps were decided by Eq. (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.

We tested the two error criteria as well as the improvement techniques described in Sects. 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 fur-

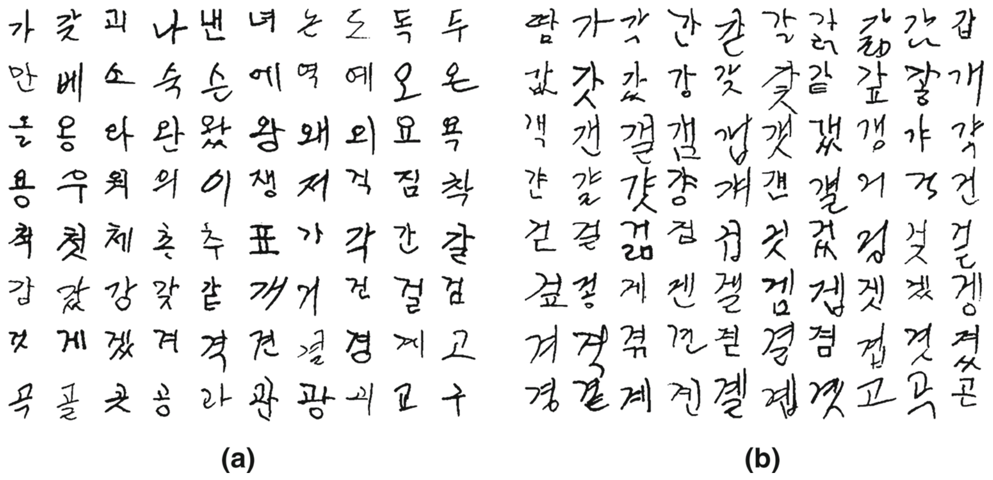

Fig. 5 Example images in SERI95a and PE92 databases

Table 5 Structure of digit recognizer (MNIST)

Layer	Type	# of feature maps	Feature map size	Window size	Stride	# of parameters
C_1	Convolution	32	28 × 28	5 × 5	1	832
P_2	Max-pooling	32	14 × 14	2 × 2	2	0
C_3	Convolution	32	10 × 10	5 × 5	1	25,632
P_4	Max-pooling	32	5 × 5	2 × 2	2	0
C_5	Convolution	256	1 × 1	5 × 5	1	205,056
F_6	Fully connected	256	1 × 1	N/A	N/A	65,792
F_7	Fully connected	10	1 × 1	N/A	N/A	2,570

Table 6 Digit recognition results

	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

ther 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.

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 very slow. 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 64 × 64 image that consists of a 60 × 60 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 on the highest two

Table 7 Structure of Hangul recognizer (SERI95a)

Layers	Type	# of feature maps	Feature map size	Window size	Stride	# of parameters
C_1	Convolution	32	60×60	5×5	1	832
P_2	Max-pooling	32	30×30	2×2	2	0
C_3	Convolution	64	26×26	5×5	1	51,264
P_4	Max-pooling	64	13×13	2×2	2	0
C_5	Convolution	128	10×10	4×4	1	131,200
P_6	Max-pooling	128	5×5	2×2	2	0
C_7	Convolution	256	2×2	4×4	1	524,544
P_8	Max-pooling	256	1×1	2×2	1	0
F_9	Fully connected	384	1×1	N/A	N/A	98,688
F_{10}	Fully connected	520	1×1	N/A	N/A	200,200

Table 8 Structure of Hangul recognizer (PE92)

Layers	Type	# of feature maps	Feature map size	Window size	Stride	# of parameters
C_1	Convolution	32	60×60	5×5	1	832
P_2	Max-pooling	32	30×30	2×2	2	0
C_3	Convolution	64	26×26	5×5	1	51,264
P_4	Max-pooling	64	13×13	2×2	2	0
C_5	Convolution	128	10×10	4×4	1	131,200
P_6	Max-pooling	128	5×5	2×2	2	0
C_7	Convolution	256	2×2	4×4	1	524,544
P_8	Max-pooling	256	1×1	2×2	1	0
F_9	Fully connected	512	1×1	N/A	N/A	131,584
F_{10}	Fully connected	2,350	1×1	N/A	N/A	266,760

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

Given that training Hangul recognizers requires a significant amount of time, and we know that the methods described in Sects. 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 Sect. 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.

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 to training samples improved the performance. 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 suffers more from a lack of training samples. In order to know whether the improvements are statistically significant, we carried out McNemar's test on the results on SERI95a database [32]. The p values of the improvements by cross-entropy criterion (A->B), edge operator (B->C), and elastic distortion (C->D) were 0.00, 2.45×10^{-3}, and 2.42×10^{-6}, respectively. These p values show that the improvements are statistically significant.

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. Most of the results in Table 9 are better than the conventional best performances. Particularly, our best performances are noticeably higher than the two conventional best results. Compared with the previous best records, our results yielded improvements of 2.25 % on SERI95a and 5.22 % on PE92, respectively. These improvements lead to error reduction rates of 35.71 %

Table 9 Hangul recognition results

	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 10 Recognition rates achieved on SERI95a and PE92 databases

Researchers	Recognition methods	SERI95a (%)	PE92 (%)
Kim and Kim [8]	Structural matching	86.30	82.20
Kang and Kim [2]	Structural matching	90.30	<u>87.70</u>
Jang and Kim [9]	Structural matching + Post-processing	93.40	N/A
Park et al. [1]	MQDF	<u>93.71</u>	85.99
Proposed	DCNN	95.96	92.92

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, we achieved 99.71 % recognition rate on the MNIST database.

References

1. Park, G.-R., Kim, I.-J., Liu, C.-L.: An evaluation of statistical methods in handwritten Hangul recognition. Int. J. Doc. Anal. Recognit. **16**(3), 273–283 (2013)
2. Kang, K.-W., Kim, J.H.: Utilization of hierarchical, stochastic relationship modeling for Hangul character recognition. IEEE Trans. Pattern Anal. Mach. Intell. **26**(9), 1185–1196 (2004)
3. Bengio, Y.: Learning deep architectures for AI. Found. Trends Mach. Learn. **2**(1), 1–127 (2009)
4. Liu, C.-L., Yin, F., Wang, Q.-F., Wang, D.-H.: ICDAR 2011 Chinese handwriting recognition competition (2011). http://www.nlpr.ia.ac.cn/events/HRcompetition/Report.html
5. Cireşan, D.C., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: IEEE Conference on Computer Vision and Pattern Recognition (2012)
6. Cireşan, D.C., Schmidhuber, J.: 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. Kim, H.Y., Kim, J.H.: Hierarchical random graph representation of handwritten characters and its application to Hangul recognition. Pattern Recognit. **34**(2), 187–201 (2001)
9. Jang, S.I.: Post-Processing of Handwritten Hangul Recognition Using Pair-Wise Grapheme Discrimination, Master Thesis, KAIST (2002)
10. Kim, D.-I., Lee, S.-W.: Automatic evaluation of handwriting qualities of handwritten hangul image database, KU-1. In: Proceedings of the 6th IWFHR, pp. 455–464, Taejon, Korea (1998)
11. Kim, D.H., Hwang, Y.S., Park, S.T., Kim, E.J., Paek, S.H., Bang, S.Y.: Handwritten Korean character image database PE92. In: Proceedings of the 2nd ICDAR, pp. 470–473 (1993)
12. Bae, H.J., Yun, J.M., Cha, E.Y.: Neural network for hand-written character recognition using dynamic bar method. In: Proceedings of the Korea Information Science Autumn Conference, vol. 17, issue 2, pp. 251–254 (1990)
13. Kim, M.W., Jang, J.S., Lim, C.D., Song, Y.S., Kim, J.H.: 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. Jeong, S.H.: Handwritten Hangul Recognition Based on Character Cluster Segmentation, Technical Memo. Electronics and Telecommunication Research Institute, Taejon, Korea (2002)
15. Torralba, A., Fergus, R., Freeman, W.: 80 million tiny images: a large dataset for non-parametric object and scene recognition. IEEE Trans. Pattern Anal. Mach. Intell. **30**(11), 1958–1970 (2008)
16. Hinton, G.E., Osindero, S., Teh, Y.: A fast learning algorithm for deep belief nets. Neural Comput. **18**(7), 1527–1554 (2006)
17. Hinton, G.E., Dayan, P., Frey, B.J., Neal, R.: The wake-sleep algorithm for self-organizing neural networks. Science **268**, 1158–1161 (1995)
18. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.-A.: Extracting and composing robust features with denoising autoencoders. In: Cohen, W.W., McCallum, A., Roweis, S.T. (eds.) Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML'08), pp. 1096–1103, ACM (2008)
19. Fukushima, K.: Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybern. **36**(4), 193–202 (1980)
20. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
21. Simard, D., Steinkraus, P.Y., Platt, J.C.: Best practices for convolutional neural networks. In: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'03), p. 958. IEEE Computer Society, Washington, DC, USA (2003)
22. Boureau, Y., Ponce, J., LeCun, Y.: A theoretical analysis of feature pooling in vision algorithms. In: Proceedings of the International Conference on Machine Learning (ICML'10) (2010)
23. http://en.wikipedia.org/wiki/Sigmoid_function
24. Ozkan, C., Erbek, F.: A comparison of activation functions for multispectral Landsat TM image classification. Photogramm. Eng. Remote Sens. **69**(11), 1225–1234 (2003)
25. http://en.wikipedia.org/wiki/Softmax_activation_function
26. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10) (2010)
27. Glorot, X., Antoine, B., Bengio, Y.: Deep Sparse Rectifier Networks. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP, vol. 15 (2011)
28. Goodhill, G., Barrow, H.: The role of weight normalization in competitive learning. Neural Comput. **6**(2), 255–269 (1994)
29. Liu, H., Ding, X.: Handwritten character recognition using gradient feature and quadratic classifier with multiple discrimination schemes. In: Proceedings. 8th International Conference on Document Analysis and Recognition, IEEE (2005)
30. Cireşan, D., et al.: Deep Big Multilayer Perceptrons for Digit Recognition, Neural Networks: Tricks of the Trade, pp. 581–598. Springer, Berlin Heidelberg (2012)
31. http://yann.lecun.com/exdb/mnist
32. Everitt, B.: The Analysis of Contingency Tables, 2nd edn. Chapman and Hall/CRC, London (1992)