Joint architecture and knowledge distillation in CNN for Chinese text recognition

Zi-Rui Wang, Jun Du*

National Engineering Laboratory for Speech and Language Information Processing, University of Science and Technology of China, Hefei, Anhui, PR China

ARTICLE INFO

ABSTRACT

The distillation technique helps transform cumbersome neural networks into compact networks so that models can be deployed on alternative hardware devices. The main advantage of distillation-based approaches include a simple training process, supported by most off-the-shelf deep learning software and no special hardware requirements. In this paper, we propose a guideline for distilling the architecture and knowledge of pretrained standard CNNs. The proposed algorithm is first verified on a large-scale task: offline handwritten Chinese text recognition (HCTR). Compared with the CNN in the state-of-the-art system, the reconstructed compact CNN can reduce the computational cost by $> 10\times$ and the model size by $> 8\times$ with negligible accuracy loss. Then, by conducting experiments on two additional classification task datasets: *Chinese Text in the Wild* (CTW) and MNIST, we demonstrate that the proposed approach can also be successfully applied on mainstream backbone networks.

© 2020 Elsevier Ltd. All rights reserved.

1. Introduction

Convolutional neural networks (CNNs) play an important role in the new wave of artificial intelligence. Since the first-generation CNNs were proposed by LeCun [1,2] for handwritten character recognition, numerous CNNs have been emerging in different applications, such as (Alex, VGG, GoogLe, Res, and Dense)-Nets in natural image recognition [3–7], DCNN in offline handwritten Chinese text recognition (HCTR) [8,9], HCCR-CNN in handwritten Chinese character recognition (HCCR) [10,11], FaceNet in face recognition [12] and FCN in speech emotion recognition [13]. Such CNNs share the same basic components, i.e., convolutional layer, pooling layer and fully-connected layer. Although these networks have dramatically improved performance and CNN-based approaches have become mainstream in a wide range of pattern recognition tasks, the trend of going deeper and wider for CNNs makes them difficult to be deployed on resource-limited devices e.g., mobile phones and embedded chips. Moreover, there is an evident fact that the current state-of-the-art CNNs still mainly depend on massive hand-crafted trail-and-error experiments. Both the architecture and the internal knowledge of a CNN should be valuable information for acceleration and compression algorithms. Accordingly, in this pa-per, we focus on both architecture and knowledge distillation in pretrained standard CNNs.

The concept of knowledge distillation, which can be traced back to Caruana's research in 2006 [14], is to transfer the knowledge from cumbersome models into smaller model. Different from knowledge distillation, the research on architecture distillation in CNN focuses on inventing new efficient convolutions or units to directly replace standard convolutions of baseline CNN. A representative work was conducted in [15] where the authors reconstructed a lightweight CNN by using multiple efficient compact blocks according to the different locations of the baseline CNN. The realization of distillation can be figuratively described as teacher-student learning in which a network with massive parameters and high performance acts as a teacher and the compressed network is a student. Both the architecture and the internal knowledge of the teacher network should be learned by the student network. The single consideration usually leads to a contradiction between optimal performance and satisfactory compression. Unlike previous distillation algorithms, in this paper, we propose a guideline for distilling the architecture and knowledge of a pretrained CNN. Specifically, instead of using multiple acceleration blocks [15], we develop a uniform block named the parsimonious convolution (ParConv) block that only consists of depthwise separable convolution (DSConv) [16] and pointwise convolution in a heterogeneous combination. In knowledge distillation, a new solving procedure loss (SPL) is added to further improve the performance of the student network. The solving procedure is represented by the differences in attention maps between two layers.

* Corresponding author at: National Engineering Laboratory for Speech and Language Information Processing, University of Science and Technology of China, No. 96, JinZhai Road, Hefei, Anhui, PR China.

E-mail addresses: cs211@mail.ustc.edu.cn (Z.-R. Wang), jundu@ustc.edu.cn (J. Du).

The effectiveness of the proposed algorithm is mainly demonstrated on offline handwritten Chinese text recognition (HCTR). The HCTR can be widely used in many applications, such as mail address recognition [17], bank check [18] and document recognition [19]. Although the HCTR has made great progress owing to deep learning [20], it remains a challenging problem for the following reasons: 1) the text line must be considered as a whole rather than isolated characters, 2) more than 7000 classes in common Chinese vocabulary and large-scale training samples, and 3) the unconstrained writing condition. The experiments are conducted on the ICDAR 2013 competition task of the CASIA-HWDB databse [21,22], which is one of the most popular benchmark databases. To the best of our knowledge, no acceleration and compression approaches in CNNs have been validated on the offline HCTR. Furthermore, in order to display the generalization ability of the proposed algorithm, we use the proposed method to reduce the resource consumption of the mainstream backbone networks on CTW [23] and MNIST [2].

The main contributions of this study can be summarized as follows:

- We propose a guideline for distilling the architecture and knowledge of pretrained standard CNNs for fast deployability on alternative hardware devices.
- In architecture distillation, we invent a parsimonious convolution block (ParConv) to directly replace vanilla convolution without other adjustments. Compared with LightweightNet [15] and DSConv [16], the proposed ParConv demonstrates its superiority in recognition performance, computational cost and storage overhead.
- In knowledge distillation, a new solving process loss (SPL) is added to further improve the performance of compressed CNN.
- The effectiveness of the proposed approach is first verified on offline HCTR. No study has investigated whether previous acceleration and compression algorithms are still feasible in this field.
- Compared with the baseline CNN in HCTR, our proposed joint architecture and knowledge distillation can reduce the computational cost by $> 10\times$ and model size by $> 8\times$ with negligible accuracy loss. Applying the algorithm to the mainstream backbone networks Res50 and Res18 [6] on CTW and MNIST, respectively, both of the reconstructed compact networks can obtain an obvious reduction in resource consumption. Especially, the corresponding compact network of Res18 can obtain a $> 9\times$ compression rate for both model size and computational cost with almost no decrease in accuracy.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 elaborates on the details of the proposed approach. Section 4 reports the experimental results and analyses. Finally, Section 5 concludes.

2. Related work

2.1. Acceleration and compression

Almost all acceleration and compression algorithms can be divided into five groups: low-rank decomposition [24–28], parameter pruning [10,29–33], parameter quantization [34–39], compact network design [40–44] and distillation [15,45–52]

As one of the first attempts for low-rank decompositions of filters, Denton et al. [24] proposed several decomposition designs along different dimensions. In [25], the $k \times k$ filters were decomposed into $k \times 1$ and $1 \times k$ filters. A representative work comes from Zhang et al. [26], where nonlinear units were considered in the decomposition algorithm based on the assumption that the filter response lies in a low-rank subspace. In [27,28], Tucker decomposition is used to achieve compression. Such algorithms need to be conducted layer by layer. Once a layer has been decomposed, the whole network is retrained by the backpropagation (BP) algorithm. For large-scale tasks, repeated decomposition and training are usually time consuming.

Parameter pruning is based on a reasonable idea that the low weights in a neural network are not important so that they can be safely removed. In [29,30], the weights were kept or removed by comparison with a fixed threshold. Xiao et al. [10] proposed the adaptive drop-weight (ADW) to dynamically increase the threshold. Liu et al. [31] proposed channel sparsity regularization to prune channels with small scaling factors. At almost the same time, Luo et al. [32] pruned filters based on the reconstruction error of the corresponding next layer by using a greedy algorithm. Fine-grained pruning [10,29] requires a special software/hardware accelerator. Although channel-level pruning [31,32] can be directly applied to existing software platforms, such as low-rank decomposition based algorithms, the requirement of repeated pruning and fine-tuning is time consuming for large-scale tasks. Besides, from recent research [53], the pruned architecture, rather than a set of inherited important weights, is more crucial to the efficiency in the final model. Compared with layerwise pruning, a global pruning strategy [33] might be more valuable.

For parameter quantization, by using the hash algorithm, Han et al. [34] divided network weights into different groups and used the weights in the same group to share a value. Vanhoucke et al. [35] used an 8-bit type instead of the common 32-bit floating type in the network. Courbariaux et al. [36] proposed a binarized neural network in which all weights and outputs are constrained to {1, -1}, while Li et al. [37] quantized weights into {-1, 0,1}. Such methods can save a significant number of resources. However, the approach in [34] requires additional space to store the original positions for shared weights, and low-bit approximation usually degrades network performance. Different from constraining the weights to +1 or −1, Ding et al. [38,39]replaced the multiply-accumulate operation with one shift or a constrained number of shifts and adds, which can make trade-offs between accuracy and computational consumption.

An efficient and effective network structure can save a significant amount of memory and computational cost and yield competitive performance. Many compact blocks have been invented to control the fast increase in network parameters, such as the Fire module in SqueezeNet [40] and the Inception module in GoogLeNet [54]. The basic unit in these networks still consists of canonical convolution. As one low-consumption (storage and computational cost) substitute, depthwise separable convolution (DSConv) was first introduced in [16,55] and has become a key building block in recent compact networks [41–44]. In addition, Guo et al. [56] proved that DSConv is the principal components of standard convolution and can approximate the standard convolution in closed form. Although the DSConv is far more efficient than standard convolution, consistent observations can be found in [15,16], simple replacement by using DSConv is not effective. Chollet et al. [16] scaled up depthwise separable filters so that the DSConv-based network Xception with the same number of parameters as the Inception V3 [57] can outperform Inception V3.

The first distillation in a neural network was completed by Hinton et al. in [45], i.e., knowledge distillation. In [45], the soft labels from multiple neural networks were used to guide the training of a single network. Soon after, Romero et al. [46] improved the algorithm of knowledge distillation by using the outputs of hidden layers and the soft labels from a shallow network with more parameters as hints to instruct a thin deep network. Recently, Zagoruyko et al. [47] attempted to transfer the defined attention map in convolutional layers from one network to another, inspired by the human visual experience. In [48], the flow of solution procedure (FSP) matrix is defined to measure the change in information be-
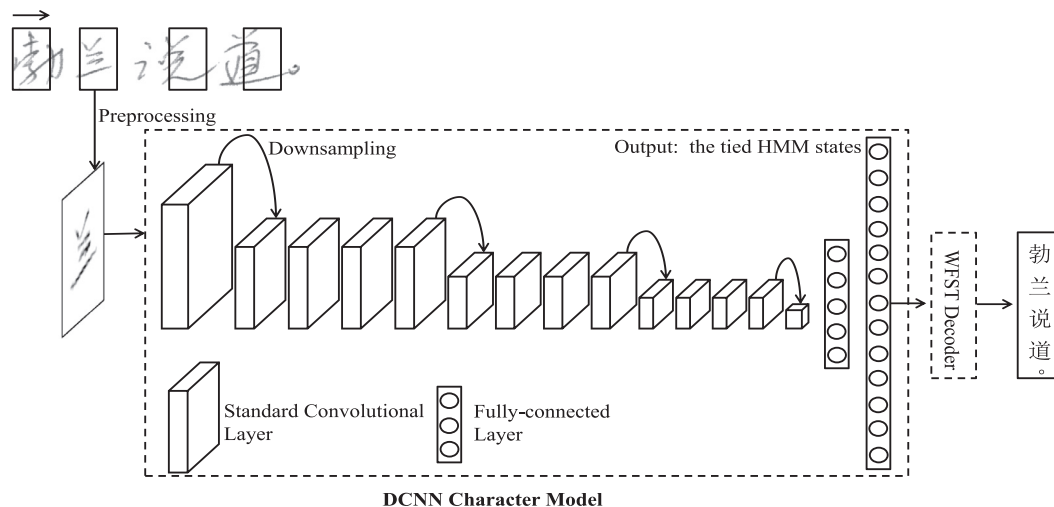
Fig. 1. Overview of our DCNN-based offline handwritten Chinese text recognition system.

tween two different layers for a compressed network to imitate the middle products of the baseline network. More recently, Liu et al. [49] integrated pixelwise loss, pairwise loss and generative adversarial loss in knowledge distillation for semantic segmentation. At almost the same time, under the framework of knowledge distillation, He et al. [50] extracted more compact middle features by using a pretrained autoencoder and proposed an affinity distillation module to capture the long-range dependency. Both of them utilize the relationship between pixels, which is important to semantic segmentation. In contrast to the diversified knowledge distillation approaches, there are not many architecture distillation [15] approaches. Recent influential works about architecture distillation can be found in [51,52]. They share a similar concept that the feature maps in the standard convolution layer are redundant. Singh et al. [51] proposed heterogeneous convolution (HetConv) with different kernel sizes in each layer to handle the corresponding parts of input feature maps while the feature maps in [52] were factorized into high frequency with fine spatial resolution and low frequency with smaller spatial size.

2.2. Offline HCTR

Offline HCTR can be formulated as a Bayesian decision problem:

$$\hat{\mathbf{C}} = \arg\max_{\mathbf{C}} p(\mathbf{C}|\mathbf{X}) \tag{1}$$

where $\mathbf{X}$ is the feature sequence of a given text line image and $\mathbf{C} = \{C_1, C_2, \ldots, C_n\}$ is the underlying n-character sequence. The research efforts for addressing such sequence modeling tasks can be divided into three categories: oversegmentation [58–60], connectionist temporal classification (CTC) [61,62] and the hidden Markov model (HMM) [8,9,63]. Almost all of these approaches benefit from the recent progress of deep learning [20]. The outputs of neural networks in different modeling methods correspond to different concepts. For example, in oversegmentation and CTC-based approaches, the outputs of the neural network are related to segmentation identification or character classes. The outputs of the network used in HMM-based approaches are posterior probabilities of states. In our recent work [9], each character is modeled by three tied states on average and a deep CNN (DCNN) with 22,080 (7360 × 3) output nodes is adopted as the character model and trained by hundreds of millions of frame-level images. As shown in Fig. 1, frame-level images are extracted from original images by a left-to-right sliding window and fed into the DCNN. Then, the posterior probabilities of states are utilized in a WFST-based decoder

[64] with/without language model (LM) for the final recognition results. In order to fit such massive training samples, the parameters of DCNN have been up to 124.5 MB and 16.02×10^8 FLOPs are needed in each inference. More details and analyses of the DCNN are shown in Section 4.

3. Architecture and knowledge distillation

Given a baseline CNN($\mathbf{W}_{fc}, \mathbf{W}_{con}$), let $\mathbf{W}_{fc}$ represents the weight set of fully-connected layers and $\mathbf{W}_{con}$ corresponds to the weight set of convolutional layers. For fully-connected layers in CNN, only the storage needs to be considered due to the relatively small computational cost. We use ℓ to denote the storage. All parameters and calculations are based on a 32-bit floating point. Assuming the number of parameters is M, the storage is computed as follows:

$$l = \left(\frac{4M}{1,024 \times 1,024}\right) \text{MB} \tag{2}$$

MB is the abbreviation for Mega Byte. The above equation is only the statistics for theoretical analysis. In experiments, the actual network storage is reported. The ratio $\gamma(\mathbf{W}_{fc}, \mathbf{W}_{con}) = \frac{\ell(\mathbf{W}_{fc})}{\ell(\mathbf{W}_{fc}) + \ell(\mathbf{W}_{con})}$ is used to measure whether a strategy π is conducted on the weights of fully-connected layers or not.

Algorithm 1 The guideline of joint architecture and knowledge distillation.

Require:
    Baseline CNN($\mathbf{W}_{fc}, \mathbf{W}_{con}$).
    Threshold T
1: Analyze the computational cost and storage overhead in baseline CNN($\mathbf{W}_{fc}, \mathbf{W}_{con}$) and compute $\gamma(\mathbf{W}_{fc}, \mathbf{W}_{con})$.
2: **if** $\gamma(\mathbf{W}_{fc}, \mathbf{W}_{con}) > T$ **then**
3:     Find a strategy $\pi(\mathbf{W}_{fc})$ to construct a new CNN($\pi(\mathbf{W}_{fc}), \mathbf{W}_{con}$) with $\gamma(\pi(\mathbf{W}_{fc}), \mathbf{W}_{con}) \leq T$ and neglected performance loss (even better).
4: **end if**
5: Build a CCNN by using ParConv blocks to replace the convolutional layers in the CNN.
6: Distill the knowledge of the CNN into the CCNN.
7: **return** The CCNN

As summarized in Algorithm 1, the guideline involves architecture distillation and knowledge distillation. We first analyze the computational cost and storage overhead of the base-
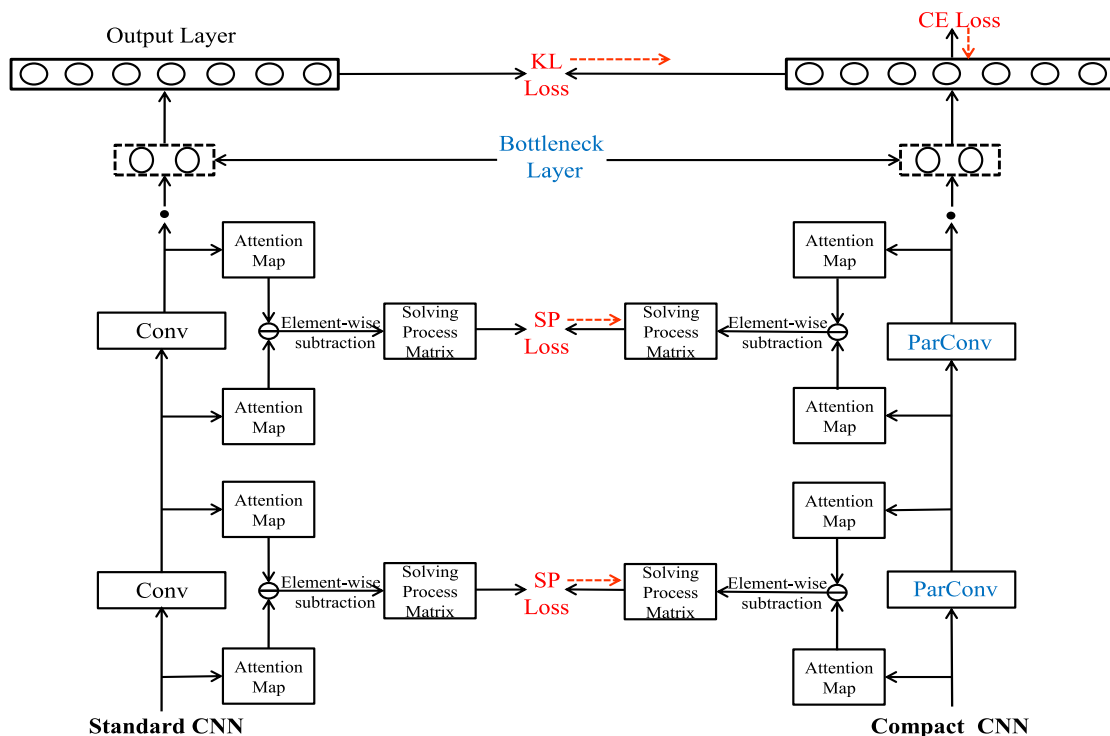
Fig. 2. The simplified framework of joint architecture and knowledge distillation.

line CNN($\mathbf{W}_{fc}$, $\mathbf{W}_{con}$) and compute the corresponding $\gamma(\mathbf{W}_{fc}, \mathbf{W}_{con})$. Then, in architecture distillation, if the weights of fully-connected layers occupy non-ignorable consumption of a certain computing resource (i.e., storage), we find a strategy $\pi(\mathbf{W}_{fc})$ to construct a new CNN($\pi(\mathbf{W}_{fc})$, $\mathbf{W}_{con}$) and ensure $\gamma(\pi(\mathbf{W}_{fc}), \mathbf{W}_{con}) \leq T$ with the neglected performance loss (even better). In most cases, it is easy to find such a strategy for fully-connected layers, e.g., global pooling [40], low-rank decomposition [65], and low-dimensional features [15]. Because we mainly focus on the compression of convolutional layers in this study, a naive solution π that depends on the number of active output targets [65] to find an appropriate bottleneck feature before the output layer is adopted. For convolutional layers, the proposed ParConv blocks are used as a direct replacement to build a compact CNN (CCNN). Finally, in order to maintain the performance of the CCNN, knowledge distillation with three kinds of losses, namely, the Kullback-Leibler (KL) divergence loss, the cross entropy (CE) loss and solving process (SP) loss, is adopted to transfer knowledge from the standard CNN into the ParConv-based CCNN. Fig. 2 illustrates the proposed algorithm. More details of the respective parts are described in the following subsections.

3.1. Bottleneck feature

As shown in Fig. 3, if there are M-dimensional features from the last conv layer, B-dimensional bottleneck features and O output nodes, the total computational costs (FLOPs) of fully-connected layers (FCs) is computed as :

$$FL_{FCs} = M \times B + B \times O \qquad (3)$$

From the above equation, we can observe that the FLOPs in fully-connected layers can be controlled by adjusting the dimension of the bottleneck feature. The FLOPs of fully-connected layers becomes smaller with the reduction of the dimension B.
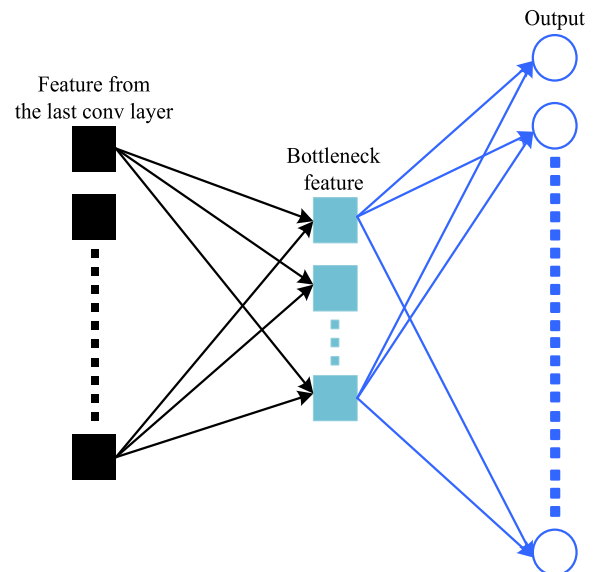

Fig. 3. Using a bottleneck layer to reduce the parameters of fully-connected layers.

3.2. Parsimonious convolution

In a standard convolutional layer, assuming the input is a square feature map, it can be represented by a three-dimensional tensor of size $D \times D \times C_{in}$. Here, D is the spatial width and height, while C_{in} is the number of input channels. Usually, (e.g., 3×3 kernel size and 1 padding), the corresponding output tensor with the channels C_{out} obtained by applying the C_{out} filters of size $K \times K \times C_{in}$ has the same spatial size $D \times D$, namely, the output size is $D \times D \times C_{out}$. Therefore, the FLOPs at this layer is:

$$FL_{Conv} = D^2 \times C_{in} \times C_{out} \times K^2 \qquad (4)$$

The depthwise separable convolution (DSConv) is made up of two components: channelwise convolution and pointwise convolution. The fundamental hypothesis behind DSConv is that cross-channel correlations and spatial correlations can be decoupled. Channelwise convolution is used to capture spatial correlations and pointwise convolution is a 1×1 standard convolution that combines information from different channels. In channelwise convolution, each output channel is only associated with one input channel so that the convolutional filters are represented by a 3-D tensor $K \times K \times C_{in}$. The FLOPs of DSConv is computed as follows:

$$FL_{DSConv} = D^2 \times (C_{in} \times K^2 + C_{in} \times C_{out}) \tag{5}$$

Compared with standard convolution, DSConv is extremely efficient in building units for many compact networks [41–44]. However, directly replacing standard convolution with DSConv leads to an increase in network depth, which makes the optimization of the reconstructed network more difficult. This problem might be alleviated by using residual connections. Besides, simple replacement by using DSConv in a standard CNN causes the network to suffer performance degradation, which may be the reason that the authors in [16] had to scale up the number of filters in DSConv.

In the proposed parsimonious convolution (ParConv), the input channels are split into two branches, one with $\alpha C_{in}(0 \leq \alpha \leq 1)$ channels for DSConv and the other with $(1 - \alpha)C_{in}$ channels for pointwise convolution. And then, the output features from different branches combine together by channel-wise addition. Specifically, before DSConv, a pointwise convolution with a channel multiplier ω is added to deeply integrate the information among channels, which is important for DSConv to extract features. In order to promote the flow of information between branches, a channel shuffle operator [41] is conducted before the input feature maps are split into two branches. The channel shuffle operator first reshapes the input channel dimension into $(2, \frac{C_{in}}{2})$, transposing and then flattening it back. For simplicity, α is set to 0.5 in all ParConvs. The FLOPs of ParConv is:

$$FL_{ParConv} = D^2 \times \frac{1}{2}C_{in} \times \frac{\omega}{2}C_{in} + D^2 \times \frac{\omega}{2}C_{in} \times K^2$$
$$+ D^2 \times \frac{\omega}{2}C_{in} \times C_{out} + D^2 \times \frac{1}{2}C_{in} \times C_{out} \tag{6}$$

Fig. 4 shows the structure of the ParConv and Table 1 lists the FLOPs ratios of different compact convolutions to standard convolution. From Table 1, it can be observed clearly that the DSConv can approach K^2 times fewer computations than standard convolution. For ParConv, under the reasonable assumption that $C_{in} =$

Table 1
The FLOPs ratios of compact convolutions to standard convolution.

Type	FLOPs Ratio
DSConv	$\frac{1}{C_{out}} + \frac{1}{K^2}$
ParConv	$\frac{1}{2K^2} + \frac{\omega}{2}\left(\frac{1}{K^2} + \frac{1}{C_{out}} + \frac{C_{in}}{2C_{out}K^2}\right)$

C_{out} and $C_{out} >> K^2$, the FLOPs ratio to standard convolution can be rewritten as follows:

$$\frac{1}{2K^2} + \frac{3\omega}{4K^2} \tag{7}$$

Obviously, the computational cost can be adjusted by changing the value of the channel multiplier ω.

3.3. Knowledge distillation with multiple losses

In order to reduce the performance gap between the standard CNN and the corresponding ParConv-based compact CNN (CCNN), knowledge distillation is necessary. Three kinds of training losses are included in the process of knowledge distillation, i.e., Kullback-Leibler (KL) divergence loss, cross entropy (CE) loss and solving procedure (SP) loss. The final loss is formulated as the weighted sum of these losses:

$$l = \mu l_{KL} + \beta l_{CE} + \lambda l_{SP} \tag{8}$$

The CE loss with so-called hard labels is the most common training criterion in classification tasks and is defined as follows:

$$l_{CE} = -\sum_t \log p(s_t|\mathbf{x}_t) \tag{9}$$

where $\log p(s_t|\mathbf{x}_t)$ is the estimated posterior probability of the target class s_t from the CNN output given the input $\mathbf{x}_t$.

The KL divergence is a measure of how one probability distribution is different from another probability distribution. In our approach, it is used to compute the difference between the output distribution of standard CNN $p_S(s|\mathbf{x}_t)$ and the corresponding distribution from CCNN $p_C(s|\mathbf{x}_t)$:

$$l_{KL} = \sum_t \sum_s p_S(s|\mathbf{x}_t) \log\left(\frac{p_S(s|\mathbf{x}_t)}{p_C(s|\mathbf{x}_t)}\right)$$
$$= \sum_t \sum_s [p_S(s|\mathbf{x}_t) \log p_S(s|\mathbf{x}_t)$$
$$- p_S(s|\mathbf{x}_t) \log p_C(s|\mathbf{x}_t)] \tag{10}$$
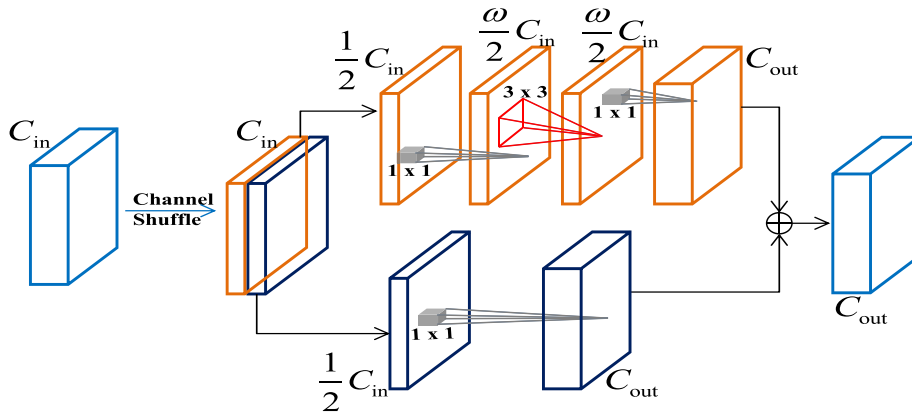
Parsimonious Convolution

Fig. 4. The structure of the ParConv. The solid cube indicates that the kernel connects with all input channels, while the rectangle with a triangle indicates the kernel only works on the corresponding single input channel.

Because we only optimize the CCNN, the KL loss in Eq. (10) can be rewritten to retain:

$$l_{KL} = -\sum_t \sum_s p_S(s|\mathbf{x}_t) \log p_C(s|\mathbf{x}_t) \tag{11}$$

Essentially, the KL loss in Eq. (11) is simplified to CE loss with soft labels. The weighted sum of KL loss and CE loss is:

$$
\begin{aligned}
\mu l_{KL} + \beta l_{CE} &= -\sum_t \sum_s \mu p_S(s|\boldsymbol{x}_t) \log p_C(s|\boldsymbol{x}_t) \\
&\quad - \sum_t \beta \log p_C(s_t|\mathbf{x}_t) \\
&= -\sum_t [(\beta + \mu p_S(s_t|\boldsymbol{x}_t)) \log p_C(s_t|\boldsymbol{x}_t) \\
&\quad + \sum_{s \neq s_t} \mu p_S(s|\boldsymbol{x}_t) \log p_C(s|\boldsymbol{x}_t)]
\end{aligned}
\tag{12}
$$

From the above formula, it is clear that the CE helps the model focus on the important parts by providing prior knowledge (ground truth of inputs).

Furthermore, it is not enough to give the CCNN the answers to problems from the standard CNN. A better teacher always explains the solving procedures of problems so that the students can handle such problems from learning one certain example. In CNN, we define a series of solving procedure matrices (SPMs). An SPM is the result of elementwise subtraction between the extracted attention feature maps from two layers, which is intuitively reasonable by using the change in outputs between two layers to represent the solving procedure. The attention map in a layer needs to emphasize valuable information for the following flow. Naturally, assuming a convolutional layer has the output tensor $\boldsymbol{O} \in R^{D \times D \times C}$ with each feature map $\boldsymbol{O}_c \in R^{D \times D}$, the attention map can be simply computed as [47]:

$$\boldsymbol{A} = \sum_{c=1}^{C} \boldsymbol{O}_c \tag{13}$$

The SPM $\boldsymbol{S}$ for the ith layer and the jth layer ($j > i$) is:

$$\boldsymbol{S} = \boldsymbol{A}_j - \boldsymbol{A}_i \tag{14}$$

Finally, the SP loss can be obtained as follows:

$$l_{SP} = \sum_t \sum_{n=1}^{N} \frac{1}{N} \times \left\| \frac{\boldsymbol{S}_{S_n}(\boldsymbol{x}_t)}{\|\boldsymbol{S}_{S_n}(\boldsymbol{x}_t)\|_F} - \frac{\boldsymbol{S}_{C_n}(\boldsymbol{x}_t)}{\|\boldsymbol{S}_{C_n}(\boldsymbol{x}_t)\|_F} \right\|_F^2 \tag{15}$$

where N is the total number of SPMs in the CNN, $\boldsymbol{S}_{S_n}$ is the nth SPM in the standard CNN, $\boldsymbol{S}_{C_n}$ is the corresponding SPM in the ParConv-based CCNN, and $\|\bullet\|_F$ is the standard Frobenius norm.

4. Experiments

The proposed distillation algorithm is mainly validated on offline handwritten Chinese text recognition (HCTR) using the CASIA database [21,22]. In addition, in order to accurately observe the performance changes of CNNs, a 5-gram LM [66] is only added in our final results. PyTorch [67] is used as a deep learning platform in all experiments.

4.1. DCNN on CASIA

The baseline DCNN architecture in [9] is adopted. Please note that except for the categories of output layers, the CNNs in [8,9] have the same architecture. According to the configuration in [9], both offline isolated handwritten Chinese character datasets (HWDB1.0, HWDB1.1 and HWDB 1.2) and the training sets of offline handwritten Chinese text datasets (HWDB2.0, HWDB2.1

and HWDB2.2) are used. In total, there are 7360 classes (Chinese characters, symbols, garbage) and 3,932,197 images. After extracting frame-level images from the original datasets, there are 148,648,249 training samples for the training of DCNN. The ICDAR 2013 competition set is adopted as the evaluation set [22]. The CER is computed as:

$$CER = \frac{N_s + N_i + N_d}{N} \tag{16}$$

where N is the total number of character samples in the evaluation set. N_s, N_i and N_d denote the number of substitution errors, insertion errors and deletion errors, respectively. In this study, we do not use additional language models because we focus on the performance of the CNN.

Each class is modeled by 3 tied HMM states on average. The input of DCNN is a normalized frame-level image of size 40×40 extracted from original images, and then each frame is extended to 48×48 by adding the margin. The output layer has $22,080(7360 \times 3)$ output nodes. In the DCNN architecture, there are 14 convolutional layers that use standard 2D convolution and are followed by batch normalization (BN) and nonlinearity activation ReLU. The number of channels continuously increases from 100 to 700. Except for the first and last convolutional layers without padding, other convolutional layers have the same padding value of 1. The stride is set to 1 for all convolutional layers, while the stride of all max pooling layers is 2 with a 3×3 window.

According to our proposed guideline in Algorithm 1, we first analyze the computational costs and storage of convolutional layers (Convs) and fully-connected layers (FCs) in DCNN. The details of the DCNN and statistical results are shown in Table 2. Here, we do not consider the consumption of the batch normalization (BN) operation and max-pooling (MaxPooling) operation because they occupy a negligible part. Based on the analyzed results, it is necessary to reduce the parameters in fully-connected layers due to a large proportion of the storage (35%). In addition, we can observe that almost all computational costs are generated by convolutional layers.

4.2. Experiments on architecture distillation

In order to make a fair comparison, the setting of hyperparameters in the training stage is the same for all experiments in this part. The minibatch size is 1000 in each iteration, the momentum is 0.9 and the weight decay is 0.0001. The learning rate is initially set to 0.1 and decreased by 0.92 after every 4000 iterations. After two epochs are conducted, the learning rate is reduced to 0.0002 and the networks are convergent. For fully-connected layers, although many algorithms can be used, we choose the simplest algorithm that controls the weights of fully-connected layers by adjusting the feature dimension before the output layer. The effectiveness of this strategy has also been reported in [15]. In Table 3, different bottleneck features are compared. The 500 dimension corresponds to the baseline DCNN. Three observations can be found. First, because the FLOPs of fully-connected layers occupy a small proportion (less than 1%), the total FLOPs remains almost unchanged when the dimension is below 100. Second, with the dimension changing from 500 to 50, the storage decreases considerably and the CER has fluctuates slightly, which means that most parameters in fully-connected layers are redundant and can be safely ignored [68]. Finally, when the dimension is smaller than 50, the storage tends to be stable, as most storage consumption is caused by convolutional layers, however, it is reasonable to observe that the CER begins to increase due to the very small number of parameters in the bottleneck feature leading to missing information. Considering the tradeoff between storage ratio γ about fully-connect layers (see Algorithm 1) and CER, we choose dimension 50 in the following experiments.

Table 2
Architecture and quantitative analysis of DCNN character model. The abbreviations f, k, s, p represent the number of feature maps, kernel size, stride length and padding size, respectively.

Layer	Configurations	Spatial Size	FLOPs ($\times 10^8$)	Fraction	Storage (MB)	Fraction
FC2	500 × 22080	1 × 1	0.1104	0.69%	42.1985	33.90%
FC1	700 × 500	1 × 1	0.0035	0.02%	1.3371	1.07%
Conv5	F:700, K:1 × 1, S:1, P:0	1 × 1	0.0049	0.03%	1.8826	1.51%
MaxPooling	K:3 × 3, S:2	-	-	-	-	-
Conv4_4	F:700, K:3 × 3, S:1, P:1	4 × 4	0.7056	4.40%	16.8362	13.52%
Conv4_3	F:700, K:3 × 3, S:1, P:1	4 × 4	0.6048	3.78%	14.4329	11.59%
Conv4_2	F:600, K:3 × 3, S:1, P:1	4 × 4	0.4320	2.70%	10.3111	8.28%
Conv4_1	F:500, K:3 × 3, S:1, P:1	4 × 4	0.3600	2.25%	8.5926	6.90%
MaxPooling	K:3 × 3, S:2	-	-	-	-	-
Conv3_4	F:500, K:3 × 3, S:1, P:1	10 × 10	2.2500	14.04%	8.5926	6.90%
Conv3_3	F:500, K:3 × 3, S:1, P:1	10 × 10	1.8000	11.23%	6.8760	5.52%
Conv3_2	F:400, K:3 × 3, S:1, P:1	10 × 10	1.0800	6.74%	4.1275	3.32%
Conv3_1	F:300, K:3 × 3, S:1, P:1	10 × 10	0.8100	5.06%	3.0956	2.49%
MaxPooling	K:3 × 3, S:2	-	-	-	-	-
Conv2_4	F:300, K:3 × 3, S:1, P:1	22 × 22	3.9204	24.47%	3.0956	2.49%
Conv2_3	F:300, K:3 × 3, S:1, P:1	22 × 22	2.6136	16.31%	2.0657	1.66%
Conv2_2	F:200, K:3 × 3, S:1, P:1	22 × 22	0.8712	5.44%	0.6905	0.55%
Conv2_1	F:100, K:3 × 3, S:1, P:1	22 × 22	0.4356	2.72%	0.3452	0.28%
MaxPooling	K:3 × 3, S:2	-	-	-	-	-
Conv1	F:100, K:3 × 3, S:1, P:0	46 × 46	0.0190	0.12%	0.0053	0.00%
Input	Frame-level image	48 × 48	-	-	-	-

Table 3
FLOPs ($\times 10^8$), STORAGE (MB), RATIO γ (%)and CER (%) comparison by using the different bottleneck features in fully-connected layers.

Low-dimensional Feature	FLOPs ($\times 10^8$)	Storage (MB)	γ (%)	CER (%)
500	16.02	124.5	34.97	9.17
100	15.96	89.74	9.78	9.04
50	15.92	85.39	5.19	9.01
25	15.91	83.22	2.71	9.38
20	15.91	82.79	2.20	9.40

Based on the DCNN with bottleneck feature 50 (DCNN_LF50), except for the initial and last convolutional layers, we replace all remaining 12 standard convolutional layers with our ParConv blocks (same channel multiplier ω for all 12 layers). The reconstructed compact CNN is notated as ParCNN_ω. For example, the ParCNN_ω0.5 indicates that the value of ω in all ParConv blocks is set to 0.5. In order to demonstrate that the proposed ParConv is a more efficient and effective replacement for standard convolution, we compare it with depthwise separable convolution (DSConv) and the architecture distillation algorithm LightweightNet proposed in [15]. Besides, we also construct the simplified ParConv (SParConv) by removing the pointwise convolution before depthwise separable convolution block in ParConv to verify the role of pointwise convolution. We directly adopt DSConv and SParConv to replace the same 12 standard convolutional layers and build the corresponding compact CNN: DSCNN and SParCNN, respectively. Table 4 lists all related results. The notation *_Res indicates that we add residual connections for all corresponding compact blocks, namely, there is another path directly connecting the input and output of the compact block.

- Comparison with different values of ω

We changed the value of ω from 0.5 to 4. As shown in Table 4, the CERs of ParCNN and ParCNN_Res consistently decrease from 10.44% to 9.59% and 10.03% to 9.53%, respectively. Naturally, the computational resources also increase with the increment of ω. Compared with the network DCNN_LF50, the ParCNN without

Table 4
The detailed results of using different compact blocks to directly replace standard convolution without knowledge distillation.

| Model	FLOPs ($\times 10^8$)	Storage (MB)	CER (%)
DCNN_LF50	15.92	89.74	9.01
DSCNN [16]	1.84	15.46	19.44
DSCNN_Res	2.67	19.79	10.01
LightweightNet [15]	2.12	23.41	10.30
SParCNN	1.81	15.30	10.45
ParCNN_ω0.5	1.56	14.14	10.44
ParCNN_ω0.5_Res	2.38	18.46	10.03
ParCNN_ω1	2.21	17.41	10.00
ParCNN_ω1_Res	3.03	21.74	9.80
ParCNN_ω2	3.50	23.95	9.72
ParCNN_ω2_Res	4.32	28.29	9.54
ParCNN_ω4	6.07	37.04	9.59
ParCNN_ω4_Res	6.90	41.37	9.53

Table 5
The results of joint architecture and knowledge distillation for DCNN.

| Model	FLOPs ($\times 10^8$)	Storage (MB)	CER (%)
DCNN	16.02	124.5	9.17
ParCNN_ω0.5	1.56	14.14	10.44
+KL			9.79
+SP			9.94
+KL+SP			9.68

Table 6
The results of the proposed approach for different acceleration and compression ratios based on channel multiplier ω.

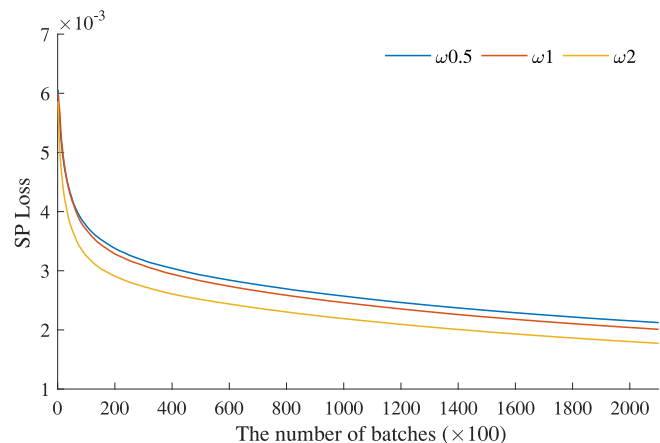| Model	FLOPs ($\times 10^8$)	Storage (MB)	Without KD	With KD
DCNN	16.02	124.5	9.17	-
ParCNN_ω0.5	1.56	14.14	10.44	9.68
ParCNN_ω1	2.21	17.41	10.00	9.37
ParCNN_ω2	3.50	23.95	9.72	9.09
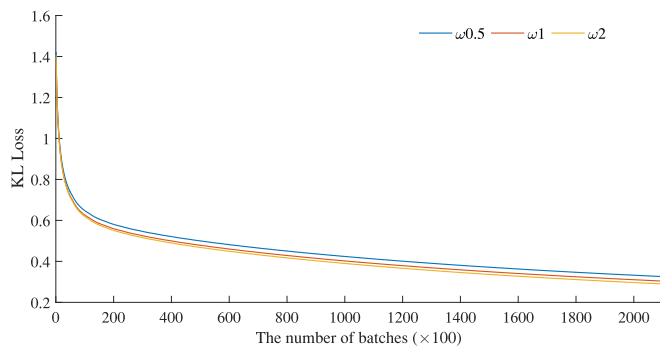
(a) The comparison of KL loss.
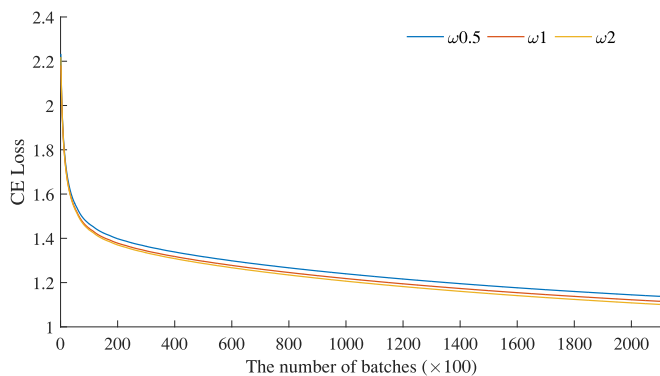
(b) The comparison of CE loss.

(c) The comparison of SP loss.

Fig. 5. The comparison of multiple losses for ParCNN_ω0.5, ParCNN_ω1 and Par-CNN_ω2 when all losses are considered simultaneously in the training stage. For simplicity, we use ω0.5, ω1, ω2 to represent respective networks in all figures.

residual connection can achieve 10.21× to 2.62×FLOPs based improvement and 6.35× to 2.42× in storage reduction, while the corresponding ParCNN_Res can reduce FLOPs from 6.69× to 2.31× and storage overhead from 4.86× to 2.17×. By comparing the results of ParCNN and ParCNN_Res, we can observe that the residual con-

nection always yields a performance improvement. Additionally, the residual connection introduces extra computational resources (approximately 0.82×10^8 FLOPs and 4.33MB) due to the pointwise convolution necessary for the special situation where the number of input and output channels of a compact block is different. Actually, in most other typical CNNs where the number of input and output channels is the same for most convolutional layers, the residual connection does not lead to too much extra consumption.

• Comparison with depthwise separable convolution

First, from the DSCNN and DSCNN_Res results, an interesting and reasonable observation can be made. Without the residual connection, the recognition performance of DSCNN significantly declines with a CER of 19.44%. In essence, directly using DSConv to replace standard convolution doubles the depth of the network, which easily leads to the degradation of network [6]. After adding the residual connection, the recognition performance of DSCNN_Res returns to a normal value (10.07%). This phenomenon reflects another advantage of our ParConv, i.e., relaxing the residual connection requirement, which can reduce the possibility of additional computations. In the case of similar CERs (10.0%), the proposed ParConv-based compact CNNs (ParCNN_ω0.5_Res, ParCNN_ω1) consume significantly fewer computing resources ($2.38 \times 10^8, 2.21 \times 10^8$ vs. 2.67×10^8 in FLOPs and 18.46MB, 17.41MB vs. 19.79MB in storage).

• Comparison with simplified parsimonious convolution

In SParConv, a quarter of the input channels are fed into depthwise separable convolution. Compared with the proposed ParCNN, all the performance indicators of SParCNN are worse than those of the proposed ParCNN_0.5, which indicates the importance of the pointwise convolution in the ParConv. Besides, the parameter ω in ParConv can be set to different values so that we can choose a compression rate that meets the performance requirements.

• Comparison with LightweightNet

We also reproduce the architecture distillation algorithm [15]. The reconstructed LightweightNet needs 2.12×10^8 FLOPs and occupies 23.41MB while the corresponding CER is 10.30%. The network ParCNN_ω0.5 with a comparable recognition performance can apparently outperform it in FLOPs and storage. Meanwhile, the networks ParCNN_ω0.5_Res and ParCNN_ω1 have similar FLOPs with LightweightNet, but lower storage and CERs.

4.3. Experiments on architecture and knowledge distillation

As shown in Table 4, a smaller value of ω can obtain a larger compression ratio but suffer from worse recognition performance. In order to reduce the performance gap between the baseline CNN and the compact CNN, it is necessary to introduce knowledge distillation. In knowledge distillation, i.e., based on Eq. (8), the weight μ is set to 0.8, β equals 0.2 and λ is 0.1. Except for the batch size set to 700, all other initial training hyper parameters are the same as the parameters in architecture distillation.

In order to excavate the best capability of the proposed approach, we first combine knowledge distillation to improve the recognition performance of the smallest network ParCNN_ω0.5. From the results of Table 5, we can observe that the knowledge

Table 7
The comparison of final results after adding the same 5-gram LM.

Model	FLOPs ($\times 10^8$)	Storage (MB)	GPU Time (ms/batch)	GPU Occupancy (MB)	CER (%)
DCNN [9]	16.02	124.5	39.3	3353	3.52
ParCNN_ω0.5	1.56	14.14	19.0	759	3.55

(a) The weight distribution of DCNN.
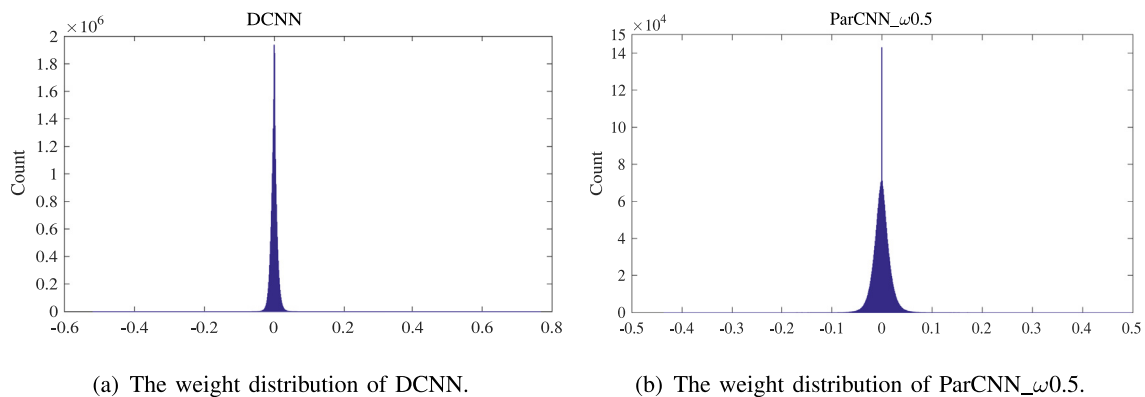

(b) The weight distribution of ParCNN_ω0.5.

Fig. 6. Weight analysis for DCNN and ParCNN_ω0.5.

distillation can yield remarkable reductions of CER: from 10.44% to 9.79% (+KL Loss), 9.94% (+SP Loss) and 9.68% (+KL Loss & +SP Loss), which demonstrates the effectiveness of the SP loss and the necessity of knowledge distillation. Compared with the baseline DCNN, our proposed joint architecture and knowledge distillation can achieve a 10×reduction in computational cost and 9×storage compression with only a 0.51% increment in CER, i.e., a relative CER increment of 5.6%.

Table 6 shows the results for different acceleration and compression ratios based on channel multiplier ω. It can be observed that the value of ωcan effectively control the acceleration and compression ratio and recognition performance. When $\omega$is set to 1, compared with the baseline DCNN, the proposed approach can reduce the computational cost and model size by > 7×with a relative CER increment of 2.2%. If we further increase the value of $\omega$to 2, the compact network ParCNN_ω2 with a > 4×acceleration ratio and > a 5×compression ratio can even obtain a better performance (9.09% vs. 9.17%).

In order to better understand why more parameters can yield better performance, we draw the learning curves of multiple losses during training for ParCNN_ω0.5, ParCNN_ω1 and ParCNN_ω2 in Fig. 5. It can be observed that all kinds of losses decrease with increasing ω,which is in line with our expectations. Another interesting observation is that the relative gap of SP loss between ParCNN_ω1 and ParCNN_ω2 is larger than other kinds of losses. This indicates that SP loss should play an important role in the training of ParCNN_ω2.

Finally, considering that LM plays an important role in HCTR, we add the same 5-gram LM [9] to compare the final results of DCNN and ParCNN_ω0.5. As shown in Table 7, it is reasonable to observe that the performance gap is almost fixed by LM, which indicates that the proposed algorithm can yield a remarkable compression ratio with negligible accuracy loss. We also test the actual runtime (milliseconds per batch) for DCNN and the proposed ParCNN_ω0.5. All models with batch size 120 are run 10 times in the same machine that is equipped with PyTorch (version 1.0.1) with GeForce RTX 2080, CUDA version 10.0.130 and CUDNN [69]version 7402. Although FLOPs reduction (theoretical) is amazing, the practical speedup (2×) is limited. The main reason is that the 1 × 1 convolutions and depthwise convolutions in PyTorch are relatively slow, and the latest CUDNN library is specially optimized for 3 × 3 convolutions. However, we observe that when running the same batch, the DCNN consumes 3353 MB of GPU memory while the ParCNN_ω0.5 only needs 759 MB.

Furthermore, in Fig. 6, it is obvious that the weights of DCNN are more concentrated around 0, which indicates massive weights in DCNN may be unimportant. The utilization of weights greatly improves in ParCNN_ω0.5, where the weight distribution becomes

Fig. 7. Some examples in the CTW dataset.

flatter. To some extent, this can explain why a small network can also achieve similar recognition accuracy. In order to demonstrate that the proposed approach can also be successfully applied on mainstream backbone networks, in the following experiments, we reconstruct the corresponding compact networks according to the structures of Res50 and Res18 and conduct experiments on CTW and MNIST respectively.

4.4. Experiments on CTW

The CTW dataset contains 1,019,402 Chinese character images extracted from 32,285 street view images. The number of Chinese character categories is 3850. These character images are annotated to different attributes: occlusion, complex background, distortion, raised character, word art and handwritten character. Examples are illustrated in Fig. 7. The image preprocessing we use is approximately consistent with the method in [23]. We first train a Res50 [6] network. As in [23], only the top 1000 frequent Chinese character categories are considered. As shown in Fig. 8, the basic unit of Res50 includes a 1 × 1 convolutional layer followed by a 3 × 3 convolutional layer, and a 1 × 1 convolutional layer in the end. The first 1 × 1 convolutional layer can form a bottleneck to reduce the total parameters. We can easily build the corresponding compact network ParRes50 by replacing the standard 3 × 3 convolution with the proposed ParConv with channel multiplier $\omega = 0.5$(see Fig. 4). In the training of Res50 and ParRes50, the minibatch size is 64, and the momentum is 0.9. The learning rate is initially set to 0.01 and decreased by 0.1 when the training loss does not improve
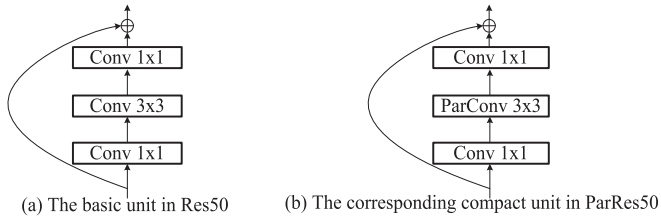
(a) The basic unit in Res50   (b) The corresponding compact unit in ParRes50

Fig. 8. The comparison of basic units in Res50 and ParRes50.

Table 8
The comparison of different networks on CTW.

Model	FLOPs ($\times 10^9$)	Storage (MB)	CER (%)
Res50 [23]	4.09	97.76	21.80
Res50 (Ours)			20.54
ParRes50	2.44	58.93	19.46

in ten consecutive observations. In Table 8, we list the recognition results and resource consumption of different networks. Considering the large number of point convolutions used in Res50, it is reasonable to observe that the reduction in parameters and FLOPs is not very significant. On the other side, we obtain the recognition accuracy improvement by simple replacement.

4.5. Experiments on MNIST

In this small dataset, according to the structure of Res18, we build the corresponding compressed network ParRes18 based on the proposed parsimonious convolution with channel multiplier $\omega = 0.5$. Fig. 9 shows the differences between Res18 and ParRes18. Then, we conduct experiments on one of the most popular datasets: the MNIST dataset that includes 60,000 training images and 10,000 test images. Each image is resized to 28×28 and labeled as a digit (0–9). We first train the three kinds of mainstream

Table 9
The overall comparison for different networks on MNIST.

Model	FLOPs ($\times 10^7$)	Storage (MB)	CER (%)
AlexNet [3]	2.15	77.57	1.02
VGG19 [4]	27.65	148.67	0.34
Res18 [6]	45.58	42.68	0.33
ParRes18	4.86	4.47	0.34

neural networks, i.e., AlexNet [3], VGG19 [4], and Res18. These network prototypes are provided by PyTorch and batch normalization is used for all convolutional layers. We use the same training criterion to train all networks: the minibatch size is 64, the momentum is 0.9, the weight decay is 0.0001 and the learning rate is set to 0.01. From Table 9, we can observe that compared with Res18, ParRes18 can obtain a $> 9\times$ acceleration ratio and compression ratio with a similar performance. Besides, it has obvious advantages over AlexNet and VGG19.

5. Discussion and conclusion

As a plug-and-play convolution block, ParConv is proposed to directly replace the standard convolution without other adjustments in the network. Unlike the Inception module in GoogLeNet that uses different kernel sizes in respective paths to extract multiscale features, the idea of the proposed ParConv derives from the opinion that the convolution filter needs not to have the same spatial correlation on all input channels in an overparameterized CNN. Therefore, in ParConv, the information can be recovered through the sum (not concatenation) of corresponding channels extracted from different kernels. In Fig. 10, a particularly simplified example where the input has four channels and the output has two channels is used to illustrate the different convolutional blocks. We can observe that the ParConv implement the heterogeneous form without additional output channels, which is different from Inception. Although one path of the proposed ParConv is similar to the in-
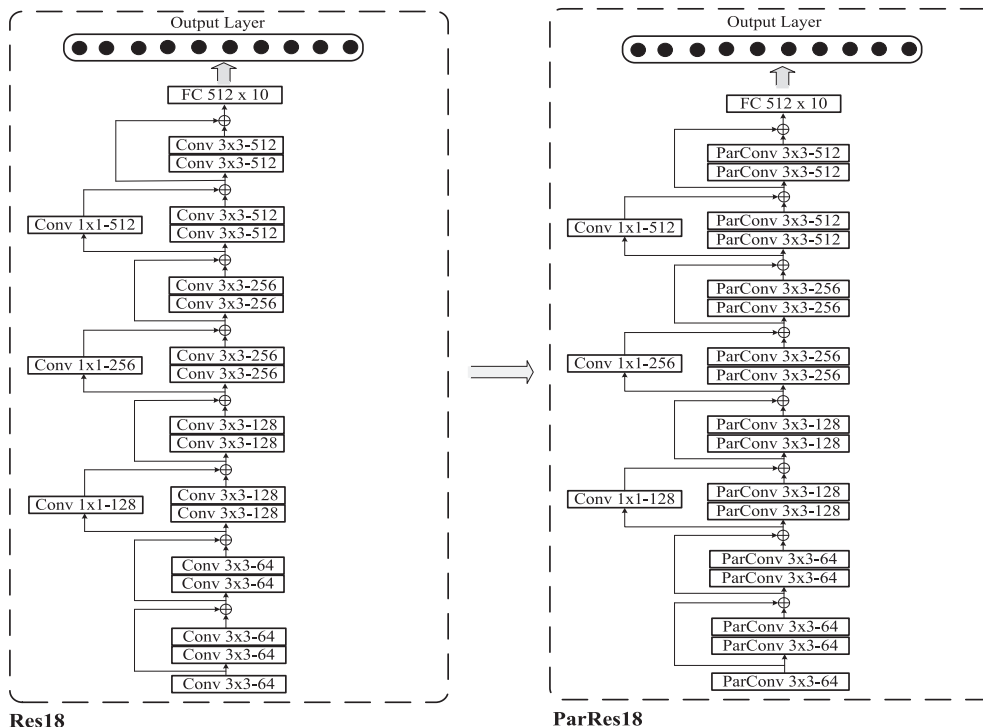

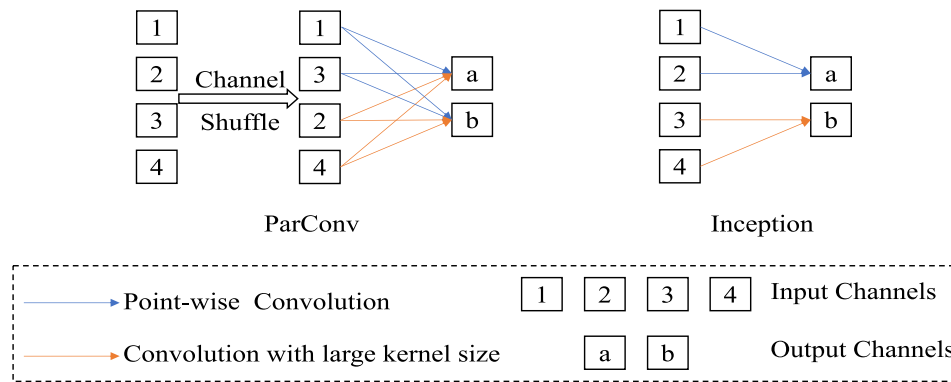**Fig. 9.** Using ParConvs to replace the Convs in Res18.

Fig. 10. A particularly simplified example about ParConv and Inception.

verted residual block in MobileNetv2 [44], they have a different starting point and use. For example, the first pointwise convolution in the inverted residual block of MobileNetv2 increases the number of input channels for extracting more abundant features. However, the ParConv is not perfect, and the accuracy drops when we attempt to achieve the highest compression rate, which is the reason why we still need knowledge distillation. As important work in the future, we will continue to develop more efficient convolution.

In conclusion, we propose a guideline for distilling the architecture and knowledge of pretrained standard CNNs. The proposed algorithm is first verified on offline handwritten Chinese text recognition. In architecture distillation, we invent a parsimonious convolution block (ParConv) to directly replace vanilla convolution without any other adjustments. To further reduce the gap between the baseline CNN and the corresponding compact CNN, knowledge distillation with multiple losses is adopted. Then, by conducting experiments on two additional classification task datasets, CTW and MNIST, we demonstrate that the proposed approach can also be successfully applied on mainstream backbone networks. However, considering that the 1×1 convolutions and depthwise convolutions in PyTorch are relatively slow and that the latest CUDNN library is specially optimized for 3×3 convolutions, the practical speedup is limited. For future work, we will combine other compression and acceleration algorithms to optimize the underlying code and complete the actual deployment.

Declaration of Competing Interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and company that could be construed as influencing the review of the manuscript entitled "Joint Architecture and Knowledge Distillation in CNN for Chinese Text Recognition".

Acknowledgment

References

[1] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Comput. 1 (4) (1989) 541–551.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al., Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.

[3] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[4] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv:1409.1556 (2014).

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.

[6] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[7] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.

[8] Z.-R. Wang, J. Du, W.-C. Wang, J.-F. Zhai, J.-S. Hu, A comprehensive study of hybrid neural network hidden Markov model for offline handwritten Chinese text recognition, Int. J. Doc. Anal.Recognit. (IJDAR) 21 (4) (2018) 241–251.

[9] Z.-R. Wang, J. Du, J.-M. Wang, Writer-aware CNN for parsimonious HMM-based offline handwritten Chinese text recognition, Pattern Recognit. (2019) 107102.

[10] X. Xiao, L. Jin, Y. Yang, W. Yang, J. Sun, T. Chang, Building fast and compact convolutional neural networks for offline handwritten Chinese character recognition, Pattern Recognit. 72 (2017) 72–81.

[11] X.-Y. Zhang, Y. Bengio, C.-L. Liu, Online and offline handwritten Chinese character recognition: a comprehensive study and new benchmark, Pattern Recognit. 61 (2017) 348–360.

[12] F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: a unified embedding for face recognition and clustering, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 815–823.

[13] Y. Zhang, J. Du, Z. Wang, J. Zhang, Y. Tu, Attention based fully convolutional network for speech emotion recognition, in: 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), IEEE, 2018, pp. 1771–1775.

[14] C. Buciluă, R. Caruana, A. Niculescu-Mizil, Model compression, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2006, pp. 535–541.

[15] T.-B. Xu, P. Yang, X.-Y. Zhang, C.-L. Liu, LightweightNet: toward fast and lightweight convolutional neural networks via architecture distillation, Pattern Recognit. 88 (2019) 272–284.

[16] F. Chollet, Exception: deep learning with depthwise separable convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1251–1258.

[17] Q. Fu, X. Ding, T. Liu, Y. Jiang, Z. Ren, A novel segmentation and recognition algorithm for Chinese handwritten address character strings, in: 18th International Conference on Pattern Recognition (ICPR'06), vol. 2, IEEE, 2006, pp. 974–977.

[18] M. Yu, P.C. Kwok, C.H. Leung, K. Tse, Segmentation and recognition of Chinese bank check amounts, Int. J. Doc. Anal. Recognit. 3 (4) (2001) 207–217.

[19] H. Fujisawa, Forty years of research in character and document recognition—an industrial perspective, Pattern Recognit. 41 (8) (2008) 2435–2446.

[20] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.

[21] C.-L. Liu, F. Yin, D.-H. Wang, Q.-F. Wang, CASIA online and offline Chinese handwriting databases, in: 2011 International Conference on Document Analysis and Recognition, IEEE, 2011, pp. 37–41.

[22] F. Yin, Q.-F. Wang, X.-Y. Zhang, C.-L. Liu, ICDAR 2013 Chinese handwriting recognition competition, in: 2013 12th International Conference on Document Analysis and Recognition, IEEE, 2013, pp. 1464–1470.

[23] T.-L. Yuan, Z. Zhu, K. Xu, C.-J. Li, S.-M. Hu, Chinese text in the wild, arXiv:1803.00085 (2018).

[24] E.L. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Advances in Neural Information Processing Systems, 2014, pp. 1269–1277.

[25] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, arXiv:1405.3866 (2014).

[26] X. Zhang, J. Zou, K. He, J. Sun, Accelerating very deep convolutional networks for classification and detection, IEEE Trans. Pattern Anal. Mach. Intell. 38 (10) (2016) 1943–1955.

[27] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications, arXiv:1511.06530 (2015).

[28] H. Ding, K. Chen, Y. Yuan, M. Cai, L. Sun, S. Liang, Q. Huo, A compact CNN-DBLSTM based character model for offline handwriting recognition with Tucker decomposition, in: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, IEEE, 2017, pp. 507–512.

[29] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.

[30] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient DNNs, in: Advances In Neural Information Processing Systems, 2016, pp. 1379–1387.

[31] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2736–2744.

[32] J.-H. Luo, J. Wu, W. Lin, ThiNet: a filter level pruning method for deep neural network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5058–5066.

[33] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, E. Choi, MorphNet: fast & simple resource-constrained structure learning of deep networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1586–1595.

[34] S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv:1510.00149 (2015).

[35] V. Vanhoucke, A. Senior, M.Z. Mao, Improving the speed of neural networks on CPUs(2011).

[36] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1, arXiv:1602.02830 (2016).

[37] F. Li, B. Zhang, B. Liu, Ternary weight networks, arXiv:1605.04711 (2016).

[38] R. Ding, Z. Liu, R. Shi, D. Marculescu, R. Blanton, LightNN: filling the gap between conventional deep neural networks and binarized networks, in: Proceedings of the on Great Lakes Symposium on VLSI 2017, ACM, 2017, pp. 35–40.

[39] R. Ding, Z. Liu, T.-W. Chin, D. Marculescu, et al., FLightNNs: lightweight quantized deep neural networks for fast and accurate inference, arXiv:1904.02835 (2019).

[40] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, SqueezeNet: alexnet-level accuracy with 50x fewer parameters and <0.5 MB model size, arXiv:1602.07360 (2016).

[41] X. Zhang, X. Zhou, M. Lin, J. Sun, ShuffleNet: an extremely efficient convolutional neural network for mobile devices, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856.

[42] N. Ma, X. Zhang, H.-T. Zheng, J. Sun, ShuffleNet v2: practical guidelines for efficient CNN architecture design, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 116–131.

[43] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: efficient convolutional neural networks for mobile vision applications, arXiv:1704.04861 (2017).

[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetv2: inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[45] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv:1503.02531 (2015).

[46] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, FitNets: hints for thin deep nets, arXiv:1412.6550 (2014).

[47] S. Zagoruyko, N. Komodakis, Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer, arXiv:1612.03928 (2016).

[48] J. Yim, D. Joo, J. Bae, J. Kim, A gift from knowledge distillation: fast optimization, network minimization and transfer learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4133–4141.

[49] Y. Liu, K. Chen, C. Liu, Z. Qin, Z. Luo, J. Wang, Structured knowledge distillation for semantic segmentation, arXiv:1903.04197 (2019).

[50] T. He, C. Shen, Z. Tian, D. Gong, C. Sun, Y. Yan, Knowledge adaptation for efficient semantic segmentation, arXiv:1903.04688 (2019).

[51] P. Singh, V.K. Verma, P. Rai, V.P. Namboodiri, HetConv: heterogeneous kernel-based convolutions for deep CNNs, arXiv:1903.04120 (2019).

[52] Y. Chen, H. Fang, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, J. Feng, Drop an octave: reducing spatial redundancy in convolutional neural networks with octave convolution, arXiv:1904.05049 (2019).

[53] Z. Liu, M. Sun, T. Zhou, G. Huang, T. Darrell, Rethinking the value of network pruning, arXiv:1810.05270 (2018).

[54] C. Szegedy, S. Ioffe, V. Vanhoucke, A.A. Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[55] L. Sifre, S. Mallat, Rigid-motion scattering for image classification, PhD thesis, Ph. D. thesis 1 (2014) 3.

[56] J. Guo, Y. Li, W. Lin, Y. Chen, J. Li, Network decoupling: from regular to depthwise separable convolutions, arXiv:1808.05517 (2018).

[57] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

[58] Q.-F. Wang, F. Yin, C.-L. Liu, Handwritten Chinese text recognition by integrating multiple contexts, IEEE Trans. Pattern Anal. Mach. Intell. 34 (8) (2012) 1469–1481.

[59] S. Wang, L. Chen, L. Xu, W. Fan, J. Sun, S. Naoi, Deep knowledge training and heterogeneous CNN for handwritten Chinese text recognition, in: 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), IEEE, 2016, pp. 84–89.

[60] Y.-C. Wu, F. Yin, C.-L. Liu, Improving handwritten Chinese text recognition using neural network language models and convolutional neural network shape models, Pattern Recognit. 65 (2017) 251–264.

[61] R. Messina, J. Louradour, Segmentation-free handwritten Chinese text recognition with LSTM-RNN, in: 2015 13th International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2015, pp. 171–175.

[62] Y.-C. Wu, F. Yin, Z. Chen, C.-L. Liu, Handwritten Chinese text recognition using separable multi-dimensional recurrent neural network, in: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, IEEE, 2017, pp. 79–84.

[63] T.-H. Su, T.-W. Zhang, D.-J. Guan, H.-J. Huang, Off-line recognition of realistic Chinese handwriting using segmentation-free strategy, Pattern Recognit. 42 (1) (2009) 167–182.

[64] M. Mohri, F. Pereira, M. Riley, Weighted finite-state transducers in speech recognition, Comput. Speech Lang. 16 (1) (2002) 69–88.

[65] T.N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, B. Ramabhadran, Low-rank matrix factorization for deep neural network training with high-dimensional output targets, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, pp. 6655–6659.

[66] S. Katz, Estimation of probabilities from sparse data for the language model component of a speech recognizer, IEEE Trans. Acoust. 35 (3) (1987) 400–401.

[67] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in PyTorch(2017).

[68] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, et al., Predicting parameters in deep learning, in: Advances in Neural Information Processing Systems, 2013, pp. 2148–2156.

[69] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, cuDNN: efficient primitives for deep learning, arXiv:1410.0759 (2014).

Zi-Rui Wang received B.Eng. and Ph.D. degrees from the Department of Electronic Engineering and Information Science, University of Science and Technology of China (USTC), in 2015 and 2020, respectively. His current research area includes artificial intelligence, deep learning and handwritten Chinese text recognition. Jun Du recei

Jun Du received B.Eng. and Ph.D. degrees from the Department of Electronic Engineering and Information Science, University of Science and Technology of China (USTC), in 2004 and 2009, respectively. From 2004 to 2009, he was with the iFlytek Speech Lab of USTC. During the above period, he worked as an Intern twice for 9 months at Microsoft Research Asia (MSRA), Beijing. In 2007, he also worked as a Research Assistant for 6 months in the Department of Computer Science at the University of Hong Kong. From July 2009 to June 2010, he worked at iFlytek Research on speech recognition. From July 2010 to January 2013, he joined MSRA as an Associate Researcher, working on handwriting recognition, OCR and speech recognition. Since February 2013, he has been with the National Engineering Laboratory for Speech and Language Information Processing (NEL-SLIP) of USTC.