2018 24th International Conference on Pattern Recognition (ICPR)
Beijing, China, August 20-24, 2018

Handwritten Digit String Recognition using Convolutional Neural Network

Hongjian Zhan, Shujing Lyu, Yue Lu
Shanghai Key Laboratory of Multidimensional Information Processing
Department of Computer Science and Technology
East China Normal University, Shanghai 200062, China
hjzhan@stu.ecnu.edu.cn, {sjlv, ylu}@cs.ecnu.edu.cn

***Abstract*—String recognition is one of the most important tasks in computer vision applications. Recently the combinations of convolutional neural network (CNN) and recurrent neural network (RNN) have been widely applied to deal with the issue of string recognition. However RNNs are not only hard to train but also time-consuming. In this paper, we propose a new architecture which is based on CNN only, and apply it to handwritten digit string recognition (HDSR). This network is composed of three parts from bottom to top: feature extraction layers, feature dimension transposition layers and an output layer. Motivated by its super performance of DenseNet, we utilize dense blocks to conduct feature extraction. At the top of the network, a CTC (connectionist temporal classification) output layer is used to calculate the loss and decode the feature sequence, while some feature dimension transposition layers are applied to connect feature extraction and output layer. The experiments have demonstrated that, compared to other methods, the proposed method obtains significant improvements on ORAND-CAR-A and ORAND-CAR-B datasets with recognition rates 92.2% and 94.02%, respectively.**

I. Introduction

Handwritten digit string recognition (HDSR) is an important topic in the area of sequence labelling. Traditional methods often segment an input image into pieces, then combine the recognition results of these pieces with path-search algorithms to get global optimal results [1]–[3]. However, there are still many problems such as various handwritten styles, connected characters and background noises when using these methods in practice.

An alternative to handle string recognition task is segmentation-free methods using recurrent neural networks (RNNs). RNN is an important branch of deep neural network, which is mainly designed for handling sequence. There are two main approaches, one unites attention-base encoder-decoder framework and the other utilizes connectionist temporal classification [4] (CTC). Shi et al. [5] constructed an attention-based model for the sequence recognition network that recognized text in a sequence recognition approach. Cheng et al. [6] applied focusing attention network with attention mechanism to address "attention drift" problem for text recognition in natural images. Benefitting from the ability of modelling the alignment between inputs and labels directly, CTC is specifically suitable for sequence classification tasks, such as speech and image string recognition. CTC is often used as an output layer for recurrent neural network. In practice, such RNN-CTC framework usually combines with a deep neural network, which generates the feature representation of input.

Nowadays with the development of deep learning, many convolutional neural network (CNN) architectures are proposed and achieve wonderful performance in most computer vision tasks [7]–[10] with the strong ability to extract features. Messina [11] firstly applied long-short time memory (LSTM) CTC network and convolutional layers to off-line Chinese handwritten text recognition. Shi et al. [12] proposed the CRNN (convolutional recurrent neural network) framework with CNN and RNN-CTC, then applied it to scene text recognition. Our previous work [13] utilized residual network and RNN-CTC to HDSR and achieved better performance than other methods.

However RNN has several drawbacks. One is that it is expensive to calculate the network output. The other is that explicit memory adds several more weights to each node, all of which must be trained. This increases the dimensionality of the problem, and potentially makes it harder to find an optimal solution. On the other hand, in many situations the input sequences are not long, and there is a lack of relations in context such as digit strings.

Some attempts using convolutional neural networks to complete sequence labeling directly without recurrent neural networks have been published. Wang et al. [14] designed a new pooling strategy named flexible Spatial Pyramid Pooling (F-SPP) to predict sequence labels directly. But the performance reported by [14] is not good enough.

In this paper, we propose a new architecture only with convolutional neural network for handwritten digit string recognition. It has several dense blocks [10] to extract feature, some dimension transposition layers to conduct dimension adjustment and at last a CTC output layer to calculate the loss in training phase and to produce the recognition results while testing.

There are several contributions of this paper. First we propose a new architecture for string recognition task that based entirely on convolutional neural network. Second, we achieve the state-of-the-art performance on the handwritten digit string recognition benchmark ORAND-CAR dataset with less computation. Last but not least, it is a general architecture for string recognition and it can be applied to other domains such as scene text recognition.

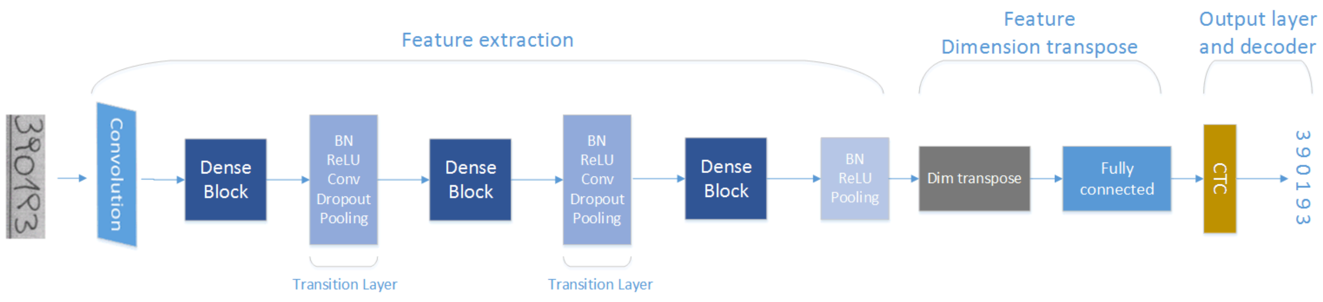


Fig. 1. The proposed architecture with three dense blocks.

The remainder of this paper is organized as follows. In Section II we present the details of the proposed network. Then, the experiments and analysis are shown in Section III. Section IV concludes this paper and discusses future work.

II. Method

The architecture proposed in this paper is shown in Fig. 1. It contains three components, feature extraction, feature dimension transposition and output layer, from bottom to top.

By taking advantage of DenseNet, we utilize dense blocks to conduct the feature extraction. Dense block consists of a group of convolutional layers with dense shortcut connections between them, which is shown in Fig. 2. These dense blocks extract a feature sequence automatically from each input image. On the top of this network a CTC output layer is adopted to calculate the loss at the training procedure and output the prediction results in testing phase. But the output of feature extraction is not suitable for CTC directly. So we add transposition layers before the features are fed into the CTC.

A. Feature Extraction

Consider a single image x_0 that is passed through a convolutional network with L layers. $H_\ell(\cdot)$ is a non-linear transformation behind each layer, where ℓ indexes the layer.

Dense Block: The main component of feature extraction is a stack of dense block. Traditional convolutional feed-forward networks connect the output of the ℓ^{th} layer as input to the $(\ell+1)^{th}$ layer [15], which gives rise to the following layer transition: $x_\ell = H_\ell(x_{\ell-1})$. ResNets add a residual connection between two non-adjacent layers that bypass the non-linear transformations with an identity function:

$$x_\ell = H_\ell(x_{\ell-1}) + x_{\ell-1}. \quad (1)$$

DenseNet embraces the observation of wonderful performance of residual connection and connects each layer to every other layer in a feed-forward fashion with residual connections. Consequently, the ℓ^{th} layer receives the feature-maps of all preceding layers, $x_0, \ldots, x_{\ell-1}$, as input:

$$x_\ell = H_\ell([x_0, x_1, \ldots, x_{\ell-1}]), \quad (2)$$

where $[x_0, x_1, \ldots, x_{\ell-1}]$ refers to the concatenation of the feature-maps produced in layers $0, \ldots, \ell-1$. In ResNet, the identity function and the output of H_ℓ are combined by summation, which may impede the information flow in the network. In dense block, the multiple input feature maps are concatenated into a single tensor. Several convolution layers with dense connectivity form a dense block. There are several parameters that control the expression of dense block: growth rate k and the number of convolution layer ℓ. The ℓ^{th} layer has $k_0 + k * (\ell - 1)$ input feature maps. k_0 is the number of channels in the input layer of this block.

Transition Layer: The name, transition layer, continues the statement in DenseNet [10]. It consists of a group of layers, as shown in Fig. 1. The batch normalization [16], ReLU [17], and dropout [18] layer are widely used to improve network's convergence speed and performance. The kernel size of the convolutional layer is $1*1$, which can reduce the number of feature maps and improve computational efficiency [9] [19]. At last, a $2*2$ average pooling layer is used to down-sampling the feature maps.

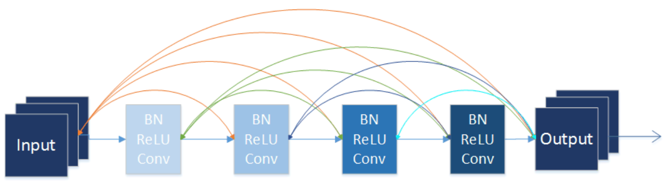


Fig. 2. A 5-layer dense block with a growth rate of k = 4.

B. Feature Dimension Transposition

The dimension of dense blocks output is not suitable for CTC directly. We need do some dimension transpositions. In some word, feature dimension transposition is also a part of feature extraction, but we show it separately to emphasize the connections between convolutional layer and CTC output layer.

The output of convolutional layer is always a 3-D tensor (4-D if we consider the batch size), i.e., the number, height and width of feature maps. But the input of CTC is a 2-D tensor. So we transpose the feature sequences into a 2-D tenser with a dimension transposition layer and a fully connected layer. Fig. 3 shows this procedure briefly.

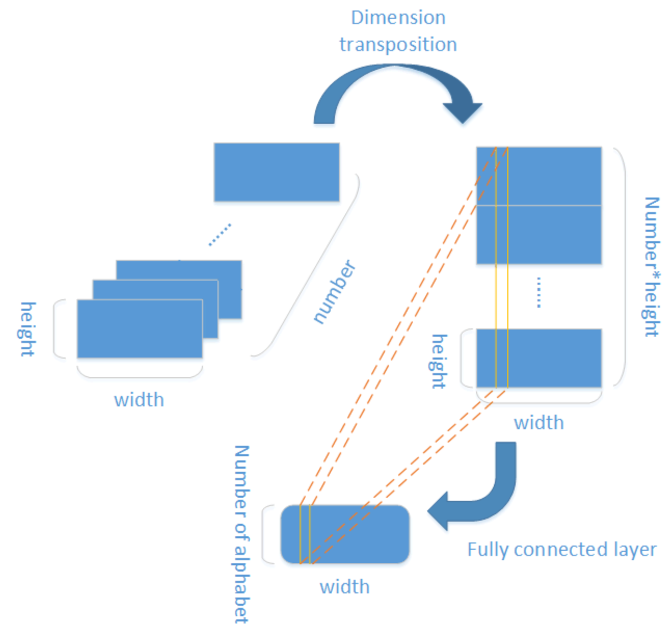


Fig. 3. A brief description of feature dimension transposition.

C. Output Layer

Connectionist temporal classification [4] is a kind of output layer. It has two main functions. One is to calculate the loss, the other is to decode the output of previous layers.

For a string recognition task, the labels are drawn from a set A (in this paper, A is the ten digits). With an extra label named *blank*, we get a new set $A^{'} = A \cup \{blank\}$, which is used in reality. The input of CTC is a sequence $y = y_1, ..., y_T$, where T is the sequence length. The corresponding labels donate as I over A. Each y_i is a probability distribution on the set $A^{'}$. We define a many-to-one function $\mathscr{F} : A^{'T} \mapsto A^{\leq T}$ to resume the repeated labels and blanks. For example $\mathscr{F}(33-9-0--1-99-3--) = 390193$. Then, a conditional probability is defined as the sum of probabilities of all π which are mapped by $\mathscr{F}$ onto I:

$$p(I|y) = \sum_{\pi \in \mathscr{F}^{-1}(I)} p(\pi|y) \tag{3}$$

where the conditional probability of π is defined as:

$$p(\pi|y) = \prod_{t=1}^{T} y_{\pi_t}^{t} \tag{4}$$

$y_{\pi_t}^{t}$ is the probability of having label π_t at timestep t. Directly computing Eq.3 is not feasible. In practice, Eq.3 is usually calculated using the forward-backward algorithm.

Donate the training dataset by $S = (\boldsymbol{X}, \boldsymbol{I})$, where X is the training image and I is the ground truth label sequence. The CTC object function $\mathscr{O}(S)$ is defined as the negative log probability of ground truth all the training examples in training set S,

$$\mathscr{O}(S) = -\sum_{(x,i)\in S} \log p(I|y) \tag{5}$$

where y is the sequence produced by the recurrent layers from x. Therefore, the network can be end-to-end trained on pairs of images and label sequences.

III. Experiments

In order to investigate the performance of the proposed method, we conduct a series of experiments on different datasets. First we train and test the network on the open datasets CVL HDS and ORAND-CAR, then we apply the method to the generated captcha image dataset. Finally we compare our method to other published approaches on these datasets.

A. Datasets

CVL HDS: Computer Vision Lab Handwritten Digit String (CVL HDS, or CVL for short) dataset consists of colored handwritten digit string images collected from about 300 writers. The CVL HDS database is composed of 7960 images, from which 1262 images have been proposed for training and the other 6698 images, for testing. The background of this dataset is clean, but the variability of writers brings high variability with respect to handwritten styles, which makes these images difficultly to handle. Some examples with different written styles are shown in Fig. 4(a).

ORAND-CAR: The ORAND-CAR consists of 11719 images obtained from the Courtesy Amount Recognition (CAR) field of real bank checks. The ORAND-CAR images come from two different sources which give the images different characteristics. These characteristics are mainly related to the image quality, kind of noise and handwriting style. Therefore, considering the two different sources, the ORAND-CAR database is split into two subsets: ORAND-CAR-A and ORAND-CAR-B, which can abbreviate as CAR-A and CAR-B. The CAR-A (Fig. 4(b)) database consists of 2009 images for training and 3784 images for testing. The CAR-B (Fig. 4(c)) database consists of 3000 training images and 2926 testing images. The string length in CAR-A is shorter than CAR-B in average.

G-Captcha: String lengths of samples in the datasets mentioned above are mostly not larger than 7. For string recognition, the longer string always means the harder task. So we create a captcha datasets by using a Python package named 'captcha'[1]. This package can generate arbitrary length captcha images with complex background, and the styles of digits are varieties, which are similar to human handwriting. The generated captcha dataset is named G-Captcha, in which string length is extended to 11. With the Python package 'captcha', we can create arbitrary lengths of captcha images. This dataset contains 14,000 images, in which 6,000 for training and 8,000

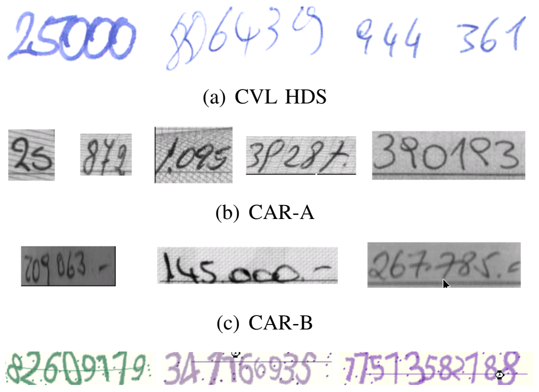
[1]https://pypi.python.org/pypi/captcha/0.1.1

TABLE I
RECOGNITION RATES OF DIFFERENT MODELS ON THE DATASETS CVL AND CAR.

Methods	CAR-A	CAR-B	CVLHDS
Tebessa I [1]	0.3705	0.2662	0.5930 -
Tebessa II [1]	0.3972	0.2772	0.6123
Singapore [1]	0.5230	0.5930	0.5040
Pernambuco [1]	0.7830	0.7543	0.5860
BeiJing [1]	0.8073	0.7013	**0.8529**
FSPP [14]	0.8261	0.8332	0.7923
CRNN [12]	0.8801	0.8979	0.2601
Saabni [3]	0.8580		-
our previous work [13]	0.8975	0.9114	0.2704
Proposed	**0.9220**	**0.9402**	0.4269

for testing. The examples of G-Captcha are shown in Fig. 4(d). Notice that, G-Captcha is the same dataset used in our pervious work [13].

The length distribution of strings the datasets is shown in Table. II and Table. IV.


(a) CVL HDS

(b) CAR-A

(c) CAR-B

(d) G-Captcha

Fig. 4. Samples of the datasets used in our experiments.

TABLE II
DISTRIBUTION OF THE CVL AND CAR DATABASES WITH RESPECT TO STRING LENGTH.

	train			test		
len	CVL	CAR-A	CAR-B	CVL	CAR-A	CAR-B
2	0	22	0	0	36	0
3	0	204	0	0	387	5
4	0	704	63	0	1425	69
5	125	903	1200	789	1475	1241
6	758	145	1599	4144	363	1452
7	379	29	137	1765	87	157
8	0	2	1	0	11	2

TABLE III
RECOGNITION RATES OF DIFFERENT MODELS ON THE G-CAPTCHA DATASET DESCRIBED ABOVE.

Methods	G-Captcha
CRNN [12]	0.9312
our previous work [13]	**0.9515**
Proposed	0.9420

TABLE IV
DISTRIBUTION OF G-CAPTCHA DATABASE WITH RESPECT TO STRING LENGTH

Length	training set	testing set
8	1500	2000
9	1500	2000
10	1500	2000
11	1500	2000

B. *Pre-Processing*

The size of images is varying in these datasets, but the fully connected layer in our network requires fixed input dimension. There are two solutions to address this issue. One is to resize the input images into a uniform size, the other is to extend the feature sequence generated by feature extractor to a fixed length. The essential of these two methods is the same. In this paper, we resize the input images to address this problem.

C. *Evaluation metrics*

Two metrics are widely used to measure string recognition performance, a hard metric and a soft metric [1]. Both of them can be defined based on the Levenshtein distance (LD), which is also known as the edit distance. The edit distance between two strings is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

In this paper, we apply the hard metric. Let a_T be a target string and a_R be the corresponding recognized string. The hard metric, or accuracy rate (AR) is computed by:

$$AR = \frac{N_a}{N} \tag{6}$$

where N is the total number of testing string images and N_a is the number of images that the $LD(a_T, a_R) = 0$. All performance indexes appear in tables of this paper are accurate rate.

D. *Experimental details*

The network is trained with ADADELTA, setting the parameter delta to 10^{-6}. On all datasets we train the network using batch size 128 for $300,000$ iterations. We also apply wight decay and momentum to train the network. The parameters of them are $5 * 10^{-4}$ and 0.9 respectively.

For dense block, we apply the memory-efficient version [20]. The implementation in Caffe can be found in this

TABLE V
INFERENCE SPEED OF DIFFERENT METHODS.

Methods	Testing Speed (ms/image)
CRNN [12]	$166.7 \sim 250.0$
our previous work [13]	$112.9 \sim 133.7$
Proposed	$23.9 \sim 35.2$

website[2]. The hyper-parameters of three dense blocks are the same. The growth rate is 8 and the number of convolution layer is 16. We set the initial channel to 128.

Our experiments are performed on a DELL workstation. The CPU is Intel Xeon E5-1650 with 3.5GHz and the GPU is NVIDIA TITAN X. The software is the latest version of Caffe [21] with cuDNN V5 accelerated on Ubuntu 14.04 LTS system.

The CRNN [12] is open source[3] with Torch [22] toolkits. We rebuild the experimental environment in our machine and apply it to HDSR for comparison.

E. Speed

One of the advantage of our method is that it is faster than RNN-containing architectures. Speed is not easy to compare, and it relies heavily on the current environment of computer. So we give a range of speed. We compare the proposed network with the RNN-containing networks such as our previous work [13] and CRNN [12].

The results are shown in Table V. We can see that without RNN, our model saves a lot of time. It is important in some real-time application scenarios.

F. Results and Analysis

Table I shows the performance results of our approach and nine existing methods on public datasets CVL and CAR datasets. The first five rows are traditional methods and others are deep learning methods. FSPP applied several convolutional layers to extract features and defined a new SPP strategy to do string recognition. Saabni also utilized deep neural networks to extract features. The network used in CRNN and our previous are similar. It can be noted as CNN-LSTM-CTC architecture. The difference between them is that our previous work applied residual convolutional layers to replace ordinary CNNs and got better performances.

From Table I we can see that our network achieves the best performance on CAR-A and CAR-B datasets than all existing methods with significant improvements. This demonstrates the effectiveness and superiority of the proposed network. We also have an obvious progress on CVL dataset than CNN-LSTM-CTC architecture. FSPP has the best performance on CVL in deep learning methods. But the network used in FSPP is not deep and it doesn't work well on CAR dataset. According to our previous analysis [13], the biggest reason of failing on CVL is the shortage of training samples. Deeper models always need larger training sets. The results on CVL show that our model has the ability to remit the shortage of training samples to some extent.

The results on G-Captcha are shown in Table III. G-Captcha is a generated dataset, we create it to verify the performance on long strings. The string length distribution of datasets can be found in Table II and Table IV. Since we can't obtain the codes of other methods, we just compare with ones that can be rebuilt in our environment. The results indicate that our model losses some advantage of handling long strings without RNN, but it still achieves a very competitive performance. Meanwhile, our method has a big progress in reducing computation.

IV. CONCLUSION

In this paper, we have presented a novel neural network for string recognition that based entirely on convolutional neural network. It avoids the drawbacks brought by RNNs. Compared to the existing RNN-containing and other methods, our approach achieves better performance on handwritten digit string recognition benchmark ORAND-CAR dataset with a remarkable advance. At the same time, it requires less computation. In the future, we plan to extent the proposed network to handwritten Chinese text recognition and other related tasks.

[2]https://github.com/Tongcheng/caffe/
[3]https://github.com/bgshih/crnn

REFERENCES

[1] M. Diem, S. Fiel, F. Kleber, R. Sablatnig, J. M. Saavedra, D. Contreras, J. M. Barrios, and L. S. Oliveira, "Icfhr 2014 competition on handwritten digit string recognition in challenging datasets (hdsrc 2014)," in *14th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2014, pp. 779–784.

[2] A. Gattal, Y. Chibani, and B. Hadjadji, "Segmentation and recognition system for unknown-length handwritten digit strings," *Pattern Analysis and Applications*, vol. 20, no. 2, pp. 307–323, 2017.

[3] R. Saabni, "Recognizing handwritten single digits and digit strings using deep architecture of neural networks," in *the 3th International Conference on Artificial Intelligence and Pattern Recognition*, 2016, pp. 1–6.

[4] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *the 23rd International Conference on Machine learning*, 2006, pp. 369–376.

[5] B. Shi, X. Wang, P. Lyu, C. Yao, and X. Bai, "Robust scene text recognition with automatic rectification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4168–4176.

[6] Z. Cheng, F. Bai, Y. Xu, G. Zheng, S. Pu, and S. Zhou, "Focusing attention: Towards accurate text recognition in natural images," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 5086–5094.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *the 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[10] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[11] R. Messina and J. Louradour, "Segmentation-free handwritten chinese text recognition with lstm-rnn," in *13th IAPR International Conference on Document Analysis and Recognition*, 2015, pp. 171–175.

[12] B. Shi, X. Bai, and C. Yao, “An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, DOI 10.1109/TPAMI.2016.2646371, 2016.

[13] H. Zhan, Q. Wang, and Y. Lu, “Handwritten digit string recognition by combination of residual network and rnn-ctc,” in *International Conference on Neural Information Processing*. Springer, 2017, pp. 583–591.

[14] Q. Wang and Y. Lu, “A sequence labeling convolutional network and its application to handwritten string recognition,” in *Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017, pp. 2950–2956.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[16] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning*, 2015, pp. 448–456.

[17] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.

[19] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[20] G. Pleiss, D. Chen, G. Huang, T. Li, L. van der Maaten, and K. Q. Weinberger, “Memory-efficient implementation of densenets,” *arXiv preprint arXiv:1707.06990*, 2017.

[21] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[22] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A matlab-like environment for machine learning,” in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.