Hindawi Publishing Corporation
Journal of Sensors
Volume 2015, Article ID 258619, 12 pages
http://dx.doi.org/10.1155/2015/258619

Research Article

Deep Convolutional Neural Networks for Hyperspectral Image Classification

Wei Hu,[1] Yangyu Huang,[1] Li Wei,[1] Fan Zhang,[1] and Hengchao Li[2,3]

[1]*College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 10029, China*
[2]*Sichuan Provincial Key Laboratory of Information Coding and Transmission, Southwest Jiaotong University, Chengdu 610031, China*
[3]*Department of Aerospace Engineering Sciences, University of Colorado, Boulder, CO 80309, USA*

Correspondence should be addressed to Wei Hu; thu.wei.hu@qq.com

Received 23 November 2014; Accepted 22 January 2015

Academic Editor: Tianfu Wu

Recently, convolutional neural networks have demonstrated excellent performance on various visual tasks, including the classification of common two-dimensional images. In this paper, deep convolutional neural networks are employed to classify hyperspectral images directly in spectral domain. More specifically, the architecture of the proposed classifier contains five layers with weights which are the input layer, the convolutional layer, the max pooling layer, the full connection layer, and the output layer. These five layers are implemented on each spectral signature to discriminate against others. Experimental results based on several hyperspectral image data sets demonstrate that the proposed method can achieve better classification performance than some traditional methods, such as support vector machines and the conventional deep learning-based methods.

1. Introduction

The hyperspectral imagery (HSI) [1] is acquired by remote sensers, which are characterized in hundreds of observation channels with high spectral resolution. Taking advantages of the rich spectral information, numerous traditional classification methods, such as k-nearest-neighbors (k-nn), minimum distance, and logistic regression [2], have been developed. Recently, some more effective feature extraction methods as well as advanced classifiers were proposed, such as spectral-spatial classification [3] and local Fisher discriminant analysis [4]. In the current literatures, support vector machine (SVM) [5, 6] has been viewed as an efficient and stable method for hyperspectral classification tasks, especially for the small training sample sizes. SVM seeks to separate two-class data by learning an optimal decision hyperplane which best separates the training samples in a kernel-included high-dimensional feature space. Some extensions of SVM in hyperspectral image classification were presented to improve the classification performance [3, 7, 8].

Neural networks (NN), such as multilayer perceptron (MLP) [9] and radial basis function (RBF) [10] neural networks, have already been investigated for classification of remote sensing data. In [11], the authors proposed a semi-supervised neural network framework for large-scale HSI classification. Actually, in remote sensing classification tasks, SVM is superior to the traditional NN in terms of classification accuracy as well as computational cost. In [12], a deeper architecture of NN has been considered a powerful model for classification, whose classification performance is competitive to SVM.

Deep learning-based methods achieve promising performance in many fields. In deep learning, the convolutional neural networks (CNNs) [12] play a dominant role for processing visual-related problems. CNNs are biologically-inspired and multilayer classes of deep learning models that use a single neural network trained end to end from raw image pixel values to classifier outputs. The idea of CNNs was firstly introduced in [13], improved in [14], and refined and simplified in [15, 16]. With the large-scale sources

of training data and efficient implementation on GPUs, CNNs have recently outperformed some other conventional methods, even human performance [17], on many vision-related tasks, including image classification [18, 19], object detection [20], scene labeling [21], house number digit classification [22], and face recognition [23]. Besides vision tasks, CNNs have been also applied to other areas, such as speech recognition [24, 25]. The technique has been verified as an effective class of models for understanding visual image content, giving some state-of-the-art results on visual image classification and other visual-related problems. In [26], the authors presented DNN for HSI classification, in which stacked autoencoders (SAEs) were employed to extract discriminative features.

CNNs have been demonstrated to provide even better classification performance than the traditional SVM classifiers [27] and the conventional deep neural networks (DNNs) [18] in visual-related area. However, since CNNs have been only considered on visual-related problems, there are rare literatures on the technique with multiple layers for HSI classification. In this paper, we have found that CNNs can be effectively employed to classify hyperspectral data after building appropriate layer architecture. According to our experiments, we observe that the typical CNNs, such as LeNet-5 [14] with two convolutional layers, are actually not applicable for hyperspectral data. Alternatively, we present a simple but effective CNN architecture containing five layers with weights for supervised HSI classification. Several experiments demonstrate excellent performance of our proposed method compared to the classic SVM and the conventional deep learning architecture. As far as we know, it is the first time to employ the CNN with multiple layers for HSI classification.

The paper is organized as follows. In Section 2, we give a brief introduction to CNNs. In Section 3, the typical CNN architecture and the corresponding training process are presented. In Section 4, we experimentally compare the performance of our method with SVM and some neural networks with different architectures. Finally, we conclude by summarizing our results in Section 5.

2. CNNs

CNNs represent feed-forward neural networks which consist of various combinations of the convolutional layers, max pooling layers, and fully connected layers and exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. Convolutional layers alternate with max pooling layers mimicking the nature of complex and simple cells in mammalian visual cortex [28]. A CNN consists of one or more pairs of convolution and max pooling layers and finally ends with a fully connected neural networks. A typical convolutional network architecture is shown in Figure 1 [24].

In ordinary deep neural networks, a neuron is connected to all neurons in the next layer. CNNs are different from ordinary NN in that neurons in convolutional layer are only sparsely connected to the neurons in the next layer, based on their relative location. That is to say, in a fully connected

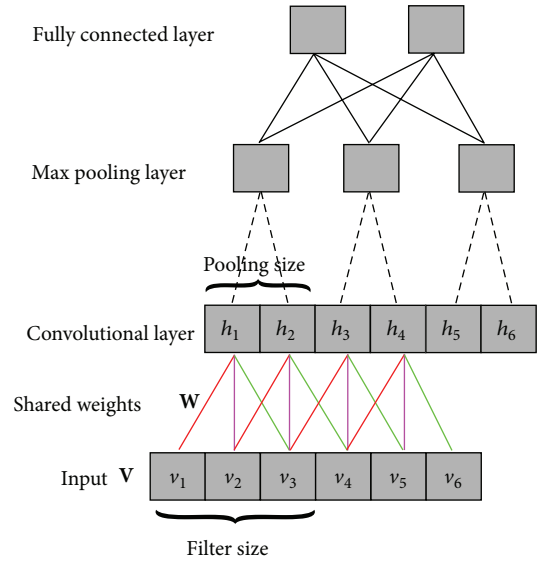

Figure 1: A typical CNN architecture consisting of a convolutional layer, a max pooling layer, and a fully connected layer.

DNNs, each hidden activation h_i is computed by multiplying the entire input **V** by weights **W** in that layer. However, in CNNs, each hidden activation is computed by multiplying a small local input against the weights **W**. The weights **W** are then shared across the entire input space, as shown in Figure 1. Neurons that belong to the same layer share the same weights. Weight sharing is a critical principle in CNNs since it helps reduce the total number of trainable parameters and leads to more efficient training and more effective model. A convolutional layer is usually followed by a max pooling layer.

Due to the replication of weights in a CNN, a feature may be detected across the input data. If an input image is shifted, the neuron detecting the feature is shifted as much. Pooling is used to make the features invariant from the location, and it summarizes the output of multiple neurons in convolutional layers through a pooling function. Typical pooling function is maximum. A max pooling function basically returns the maximum value from the input. Max pooling partitions the input data into a set of nonoverlapping windows and outputs the maximum value for each subregion and reduces the computational complexity for upper layers and provides a form of translation invariance. To be used for classification, the computation chain of a CNN ends in a fully connected network that integrates information across all locations in all the feature maps of the layer below.

Most of CNNs working in image recognition have the lower layers composed to alternate convolutional and max pooling layers, while the upper layers are fully connected traditional MLP NNs. For example, LeNet-5 is such a CNN architecture presented for handwritten digit recognition [14] firstly and then it is successfully used for solving other visual-related problems. However, LeNet-5 might not be directly employed for HSI classification, especially for small-size data sets, according to our experiments in Section 4. In this paper, we will explore what is the suitable architecture and strategy for CNN-based HSI classification.

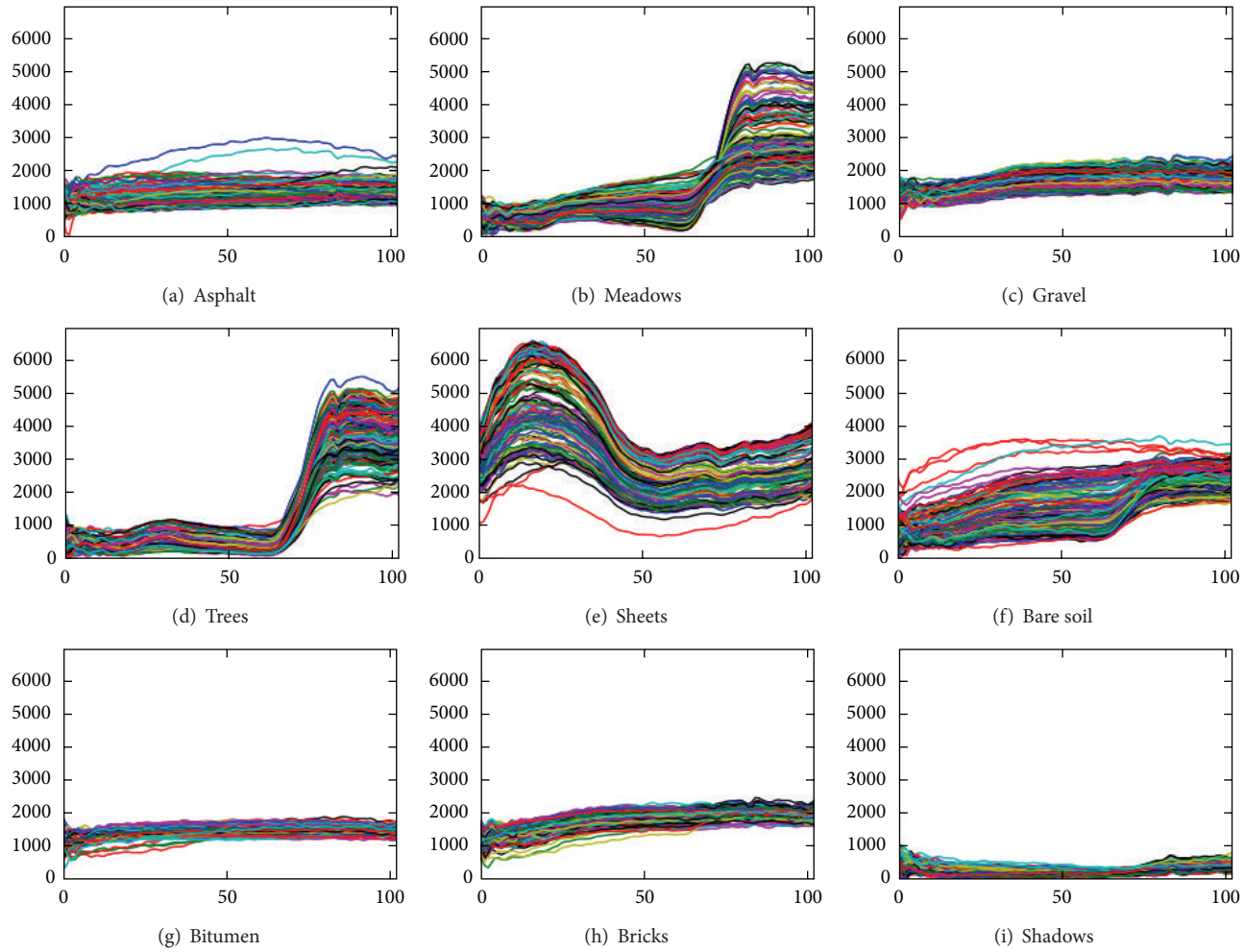

FIGURE 2: Spectral signatures of the 9 classes selected from University of Pavia data set with 103 channels/spectral bands.

3. CNN-Based HSI Classification

3.1. Applying CNNs to HSI Classification. The hierarchical architecture of CNNs is gradually proved to be the most efficient and successful way to learn visual representations. The fundamental challenge in such visual tasks is to model the intraclass appearance and shape variation of objects. The hyperspectral data with hundreds of spectral channels can be illustrated as 2D curves (1D array) as shown in Figure 2 (9 classes are selected from the University of Pavia data set). We can see that the curve of each class has its own visual shape which is different from other classes, although it is relatively difficult to distinguish some classes with human eye (e.g., gravel and self-blocking bricks). We know that CNNs can achieve competitive and even better performance than human being in some visual problems, and its capability inspires us to study the possibility of applying CNNs for HSI classification using the spectral signatures.

3.2. Architecture of the Proposed CNN Classifier. The CNN varies in how the convolutional and max pooling layers are realized and how the nets are trained. As illustrated in Figure 3, the net contains five layers with weights, including the input layer, the convolutional layer C1, the max pooling layer M2, the full connection layer F3, and the output layer. Assuming θ represents all the trainable parameters (weight

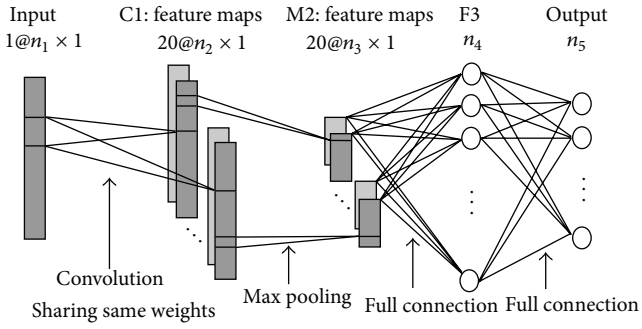

FIGURE 3: The architecture of the proposed CNN classifier. The input represents a pixel spectral vector, followed by a convolution layer and a max pooling layer in turns to compute a set of 20 feature maps classified with a fully connected network.

values), $\theta = \{\theta_i\}$ and $i = 1, 2, 3, 4$, where θ_i is the parameter set between the $(i-1)$th and the ith layer.

In HSI, each HSI pixel sample can be regarded as a 2D image whose height is equal to 1 (as 1D audio inputs in speech recognition). Therefore, the size of the input layer is just $(n_1, 1)$, and n_1 is the number of bands. The first hidden convolutional layer C1 filters the $n_1 \times 1$ input data with 20 kernels of size $k_1 \times 1$. Layer C1 contains $20 \times n_2 \times 1$ nodes, and $n_2 = n_1 - k_1 + 1$. There are $20 \times (k_1 + 1)$ trainable parameters between layer C1 and the input layer. The max pooling layer M2 is the second hidden layer, and the kernel size is $(k_2, 1)$. Layer M2 contains $20 \times n_3 \times 1$ nodes, and $n_3 = n_2/k_2$. There is no parameter in this layer. The fully connected layer F3 has n_4 nodes and there are $(20 \times n_3 + 1) \times n_4$ trainable parameters between this layer and layer M2. The output layer has n_5 nodes, and there are $(n_4 + 1) \times n_5$ trainable parameters between this layer and layer F3. Therefore, the architecture of our proposed CNN classifier totally has $20 \times (k_1 + 1) + (20 \times n_3 + 1) \times n_4 + (n_4 + 1) \times n_5$ trainable parameters.

Classifying a specified HSI pixel requires the corresponding CNN with the aforementioned parameters, where n_1 and n_5 are the spectral channel size and the number of output classes of the data set, respectively. In our experiments, k_1 is better to be $\lceil n_1/9 \rceil$, and $n_2 = n_1 - k_1 + 1$. n_3 can be any number between 30 and 40, and $k_2 = \lceil n_2/n_3 \rceil$. n_4 is set to be 100. These choices might not be the best but are effective for general HSI data.

In our architecture, layer C1 and M2 can be viewed as a trainable feature extractor to the input HSI data, and layer F3 is a trainable classifier to the feature extractor. The output of subsampling is the actual feature of the original data. In our proposed CNN structure, 20 features can be extracted from each original hyperspectral, and each feature has n_3 dimensions.

Our architecture has some similarities to architectures that applied CNN for frequency domain signal in speech recognition [24, 25]. We think it is caused by the similarity between 1D input of speech spectrum and hyperspectral data. Different from [24, 25], our network varies according to the spectral channel size and the number of output classes of input HSI data.

3.3. Training Strategies. Here, we introduce how to learn the parameter space of the proposed CNN classifier. All the trainable parameters in our CNN should be initialized to be a random value between −0.05 and 0.05. The training process contains two steps: forward propagation and back propagation. The forward propagation aims to compute the actual classification result of the input data with current parameters. The back propagation is employed to update the trainable parameters in order to make the discrepancy between the actual classification output and the desired classification output as small as possible.

3.3.1. Forward Propagation. Our $(L+1)$-layer CNN network ($L = 4$ in this work) consists of n_1 input units in layer INPUT, n_5 output units in layer OUTPUT, and several so-called hidden units in layers C2, M3, and F4. Assuming $\mathbf{x}_i$ is the input of the ith layer and the output of the $(l-1)$th layer, then we can compute $\mathbf{x}_{i+1}$ as

$$\mathbf{x}_{i+1} = f_i(\mathbf{u}_i), \tag{1}$$

where

$$\mathbf{u}_i = \mathbf{W}_i^T \mathbf{x}_i + \mathbf{b}_i, \tag{2}$$

and $\mathbf{W}_i^T$ is a weight matrix of the ith layer acting on the input data, and $\mathbf{b}_i$ is an additive bias vector for the ith layer. $f_i(\cdot)$ is the activation function of the ith layer. In our designed architecture, we choose the hyperbolic tangent function $\tanh(\mathbf{u})$ as the activation function in layer C1 and layer F3. The maximum function $\max(\mathbf{u})$ is used in layer M2. Since the proposed CNN classifier is a multiclass classifier, the output of layer F3 is fed to n_5 way softmax function which produces a distribution over the n_5 class labels, and the softmax regression model is defined as

$$\mathbf{y} = \frac{1}{\sum_{k=1}^{n_5} e^{\mathbf{W}_{L,k}^T \mathbf{x}_L + \mathbf{b}_{L,k}}} \begin{bmatrix} e^{\mathbf{W}_{L,1}^T \mathbf{x}_L + \mathbf{b}_{L,1}} \\ e^{\mathbf{W}_{L,2}^T \mathbf{x}_L + \mathbf{b}_{L,2}} \\ \vdots \\ e^{\mathbf{W}_{L,n_5}^T \mathbf{x}_L + \mathbf{b}_{L,n_5}} \end{bmatrix}. \tag{3}$$

The output vector $\mathbf{y} = \mathbf{x}_{L+1}$ of the layer OUTPUT denotes the final probability of all the classes in the current iteration.

3.3.2. Back Propagation. In the back propagation stage, the trainable parameters are updated by using the gradient descent method. It is realized by minimizing a cost function and computing the partial derivative of the cost function with respect to each trainable parameter [29]. The loss function used in this work is defined as

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n_5} 1\left\{j = \mathbf{Y}^{(i)}\right\} \log\left(\mathbf{y}_j^{(i)}\right), \tag{4}$$

where m is the number of training samples. $\mathbf{Y}$ is the desired output. $\mathbf{y}_j^{(i)}$ is the jth value of the actual output $\mathbf{y}^{(i)}$ (see (3)) of the ith training sample and is a vector whose size is n_5. In the desired output $Y^{(i)}$ of the ith sample, the probability value of the labeled class is 1, and the probability values of other classes are 0. $1\{j = \mathbf{Y}^{(i)}\}$ means, if j is equal to the desired label of the ith training sample, its value is 1; otherwise, its value is 0. We add a minus sign to the front of $J(\theta)$ in order to make the computation more convenient.

The derivative of the loss function with respect to $\mathbf{u}_i$ is

$$\delta_i = \frac{\partial J}{\partial \mathbf{u}_i} = \begin{cases} -(\mathbf{Y} - \mathbf{y}) \circ f'(\mathbf{u}_i), & i = L \\ \left(\mathbf{W}_i^T \delta_{i+1}\right) \circ f'(\mathbf{u}_i), & i < L, \end{cases} \tag{5}$$

where $\circ$ denotes element-wise multiplication. $f'(\mathbf{u}_i)$ can be easily represented as

$$f'(\mathbf{u}_i) = \begin{cases} (1 - f(\mathbf{u}_i)) \circ (1 + f(\mathbf{u}_i)), & i = 1, 3 \\ \text{null}, & i = 2 \\ f(\mathbf{u}_i) \circ (1 - f(\mathbf{u}_i)), & i = 4. \end{cases} \tag{6}$$

▷ *Constructing the CNN Model*
function INITCNNMODEL (θ, $[n_{1-5}]$)
  layerType = [convolution, max-pooling, fully-connected, fully-connected];
  layerActivation = [tanh(), max(), tanh(), softmax()]
  model = new Model();
  for i = 1 to 4 **do**
    layer = new Layer();
    layer.type = layerType[i];
    layer.inputSize = n_i
    layer.neurons = new Neuron $[n_{i+1}]$;
    layer.params = θ_i;
    model.addLayer(layer);
  end for
  return model;
end function
▷ *Training the CNN Model*
Initialize learning rate α, number of max iteration ITER_{max}, min error ERR_{min}, training batchs $\text{BATCHES}_{\text{training}}$, bach size $\text{SIZE}_{\text{batch}}$, and so on;
Compute n_2, n_3, n_4, k_1, k_2, according to n_1 and n_5;
Generate random weights θ of the CNN;
cnnModel = InitCNNModel(θ, $[n_{1-5}]$);
iter = 0; err = +inf;
while err > ERR_{min} and iter < ITER_{max} **do**
  err = 0;
  for bach = 1 to $\text{BATCHES}_{\text{training}}$ **do**
    $[\nabla_\theta J(\theta), J(\theta)]$ = cnnModel.train (TrainingDatas, TrainingLabels), as (4) and (8);
    Update θ using (7);
    err = err + mean($J(\theta)$);
  end for
  err = err/$\text{BATCHES}_{\text{training}}$;
  iter++;
end while
Save parameters θ of the CNN;

Algorithm 1: Our CNN-based method.

Therefore, on each iteration, we would perform the update

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta) \tag{7}$$

for adjusting the trainable parameters, where α is the learning factor ($\alpha = 0.01$ in our implementation), and

$$\nabla_\theta J(\theta) = \left\{ \frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \ldots, \frac{\partial J}{\partial \theta_L} \right\}. \tag{8}$$

We know that θ_i contains $\mathbf{W}_i$ and $\mathbf{b}_i$, and

$$\frac{\partial J}{\partial \theta_i} = \left\{ \frac{\partial J}{\partial \mathbf{W}_i}, \frac{\partial J}{\partial \mathbf{b}_i} \right\}, \tag{9}$$

where

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{W}_i} &= \frac{\partial J}{\partial \mathbf{u}_i} \circ \frac{\partial \mathbf{u}_i}{\partial \mathbf{W}_i} = \frac{\partial J}{\partial \mathbf{u}_i} \circ \mathbf{x}_i = \delta_i \circ \mathbf{x}_i, \\ \frac{\partial J}{\partial \mathbf{b}_i} &= \frac{\partial J}{\partial \mathbf{u}_i} \circ \frac{\partial \mathbf{u}_i}{\partial \mathbf{b}_i} = \frac{\partial J}{\partial \mathbf{u}_i} = \delta_i. \end{aligned} \tag{10}$$

With an increasing number of training iteration, the return of the cost function is smaller, which indicates that the actual output is closer to the desired output. The iteration stops when the discrepancy between them is small enough. We use average sum of squares to represent the discrepancy. Finally, the trained CNN is ready for HSI classification. The summary of the proposed algorithm is shown in Algorithm 1.

3.4. Classification. Since the architecture and all corresponding trainable parameters are specified, we can build the CNN classifier and reload saved parameters for classifying HSI data. The classification process is just like the forward propagation step, in which we can compute the classification result as (3).

4. Experiments

All the programs are implemented using Python language and Theano [30] library. Theano is a Python library that makes us easily define, optimize, and evaluate mathematical expressions involving multidimensional arrays efficiently and conveniently on GPUs. The results are generated on a PC

Table 1: Number of training and test samples used in the Indian Pines data set.

Number	Class	Training	Test
1	Corn-notill	200	1228
2	Corn-mintill	200	630
3	Grass-pasture	200	283
4	Hay-windrowed	200	278
5	Soybean-notill	200	772
6	Soybean-mintill	200	2255
7	Soybean-clean	200	393
8	Woods	200	1065
	Total	1600	6904

equipped with an Intel Core i7 with 2.8 GHz and Nvidia GeForce GTX 465 graphics card.

4.1. The Data Sets. Three hyperspectral data, including Indian Pines, Salinas, and University of Pavia scenes, are employed to evaluate the effectiveness of the proposed method. For all the data, we randomly select 200 labeled pixels per class for training and all other pixels in the ground truth map for test. Development data are derived from the available training data by further dividing them into training and testing samples for tuning the parameters of the proposed CNN classifier. Furthermore, each pixel is scaled to $[-1.0, +1.0]$ uniformly.

The Indian Pines data set was gathered by Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) sensor in northwestern Indiana. There are 220 spectral channels in the 0.4 to 2.45 μm region of the visible and infrared spectrum with a spatial resolution of 20 m. From the statistical viewpoint, we discard some classes which only have few labeled samples and select 8 classes for which the numbers of training and testing samples are listed in Table 1. The layer parameters of this data set in the proposed CNN classifier are set as follows: $n_1 = 220$, $k_1 = 24$, $n_2 = 197$, $k_2 = 5$, $n_3 = 40$, $n_4 = 100$, and $n_5 = 8$, and the number of total trainable parameters in the data set is 81408.

The second data employed was also collected by the AVIRIS sensor, capturing an area over Salinas Valley, California, with a spatial resolution of 3.7 m. The image comprises 512×217 pixels with 220 bands. It mainly contains vegetables, bare soils, and vineyard fields (http://www.ehu.es/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes). There are also 16 different classes, and the numbers of training and testing samples are listed in Table 2. The layer parameters of this data set in our CNN are set to be $n_1 = 224$, $k_1 = 24$, $n_2 = 201$, $k_2 = 5$, $n_3 = 40$, $n_4 = 100$, and $n_5 = 16$, and the number of total trainable parameters in the data set is 82216.

The University of Pavia data set was collected by the Reflective Optics System Imaging Spectrometer (ROSIS) sensor. The image scene, with a spatial coverage of 610×340 pixels covering the city of Pavia, Italy, was collected under the HySens project managed by DLR (the German Aerospace Agency). The data set has 103 spectral bands prior to water band removal. It has a spectral coverage from 0.43 to 0.86 μm and a spatial resolution of 1.3 m. Approximately 42776 labeled pixels with 9 classes are from the ground truth map, and the numbers of training and testing samples are shown in Table 3. The layer parameters of this data set in our CNN are set to be $n_1 = 103$, $k_1 = 11$, $n_2 = 93$, $k_2 = 3$, $n_3 = 30$, $n_4 = 100$, and $n_5 = 9$, and the number of total trainable parameters in the data set is 61249.

Table 2: Number of training and test samples used in the Salinas data set.

Number	Class	Training	Test
1	Broccoli green weeds 1	200	1809
2	Broccoli green weeds 2	200	3526
3	Fallow	200	1776
4	Fallow rough plow	200	1194
5	Fallow smooth	200	2478
6	Stubble	200	3759
7	Celery	200	3379
8	Grapes untrained	200	11071
9	Soil vineyard develop	200	6003
10	Corn senesced green weeds	200	3078
11	Lettuce romaine, 4 wk	200	868
12	Lettuce romaine, 5 wk	200	1727
13	Lettuce romaine, 6 wk	200	716
14	Lettuce romaine, 7 wk	200	870
15	Vineyard untrained	200	7068
16	Vineyard vertical trellis	200	1607
	Total	3200	50929

Table 3: Number of training and test samples used in University of Pavia data set.

Number	Class	Training	Test
1	Asphalt	200	6431
2	Meadows	200	18449
3	Gravel	200	1899
4	Trees	200	2864
5	Sheets	200	1145
6	Bare soil	200	4829
7	Bitumen	200	1130
8	Bricks	200	3482
9	Shadows	200	747
	Total	1800	40976

4.2. Results and Comparisons. Table 4 provides the comparison of classification performance between the proposed method and the traditional SVM classifier. SVM with RBF kernel is implemented using the `libsvm` package (http://www.csie.ntu.edu.tw/~cjlin/libsvm); cross validation is also employed to determine the related parameters, and all optimal ones are used in following experiments. It is obvious that our proposed method has better performance

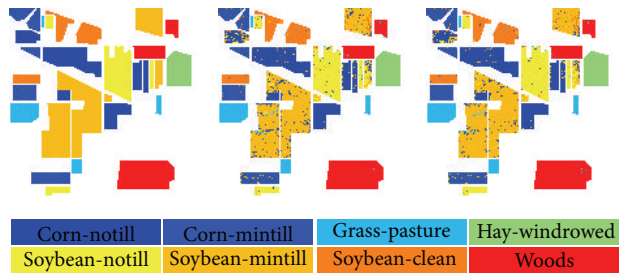

Figure 4: RGB composition maps resulting from classification for the Indian Pines data set. From left to right: ground truth, RBF-SVM, and the proposed method.

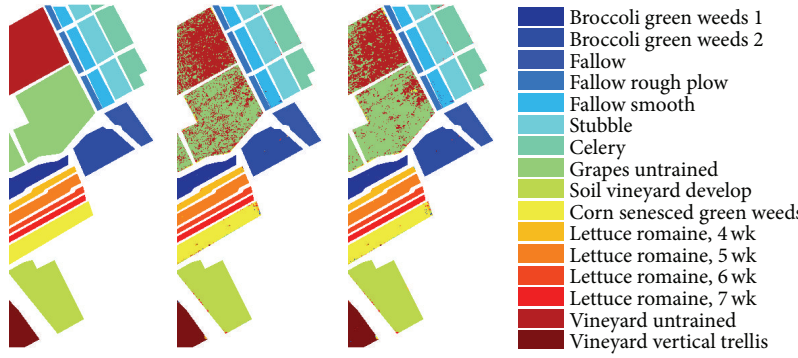

Figure 5: RGB composition maps resulting from classification for the Salinas data set. From left to right: ground truth, RBF-SVM, and the proposed method.

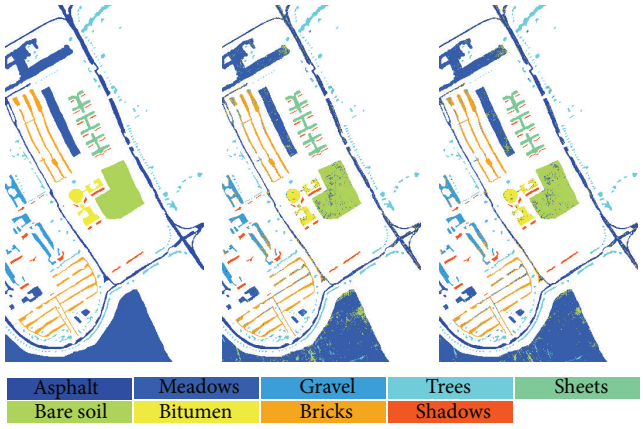

Figure 6: Thematic maps resulting from classification for University of Pavia data set. From left to right: ground truth, RBF-SVM, and the proposed method.

Table 4: Comparison of results between the proposed CNN and RBF-SVM using three data sets.

Data set	The proposed CNN	RBF-SVM
Indian Pines	90.16%	87.60%
Salinas	92.60%	91.66%
University of Pavia	92.56%	90.52%

Table 5: Results of comparison with different neural networks on the Indian Pines data set.

Method	Training time	Testing time	Accuracy
Two-layer NN	2800 s	1.65 s	86.49%
DNN	6500 s	3.21 s	87.93%
LeNet-5	5100 s	2.34 s	88.27%
Our CNN	4300 s	1.98 s	90.16%

(approximate 2% gain) than SVM classifier using all the three data sets. Figures 4, 5, and 6 illustrate the corresponding classification maps obtained with our proposed method and RBF-SVM classifier. Furthermore, compared with RBF-SVM, the proposed CNN classifier has higher classification accuracy not only for the overall data set but also for almost all the specific classes as shown in Figure 7.

Figure 8 further illustrates the relationship between classification accuracies and the training time (the test time is also included) for three experimental data sets. With the increasing of training time, the classification accuracy of each data can reach over 90%. We must admit that the training process is relatively time-consuming to achieve good performance; however, the proposed CNN classifier shares the same advantages (e.g., fast on testing) of deep learning algorithms (see Table 5). Moreover, our implementation of CNN could be improved greatly on efficiency, or we can use other CNN frameworks, such as Caffe [31], to reduce training

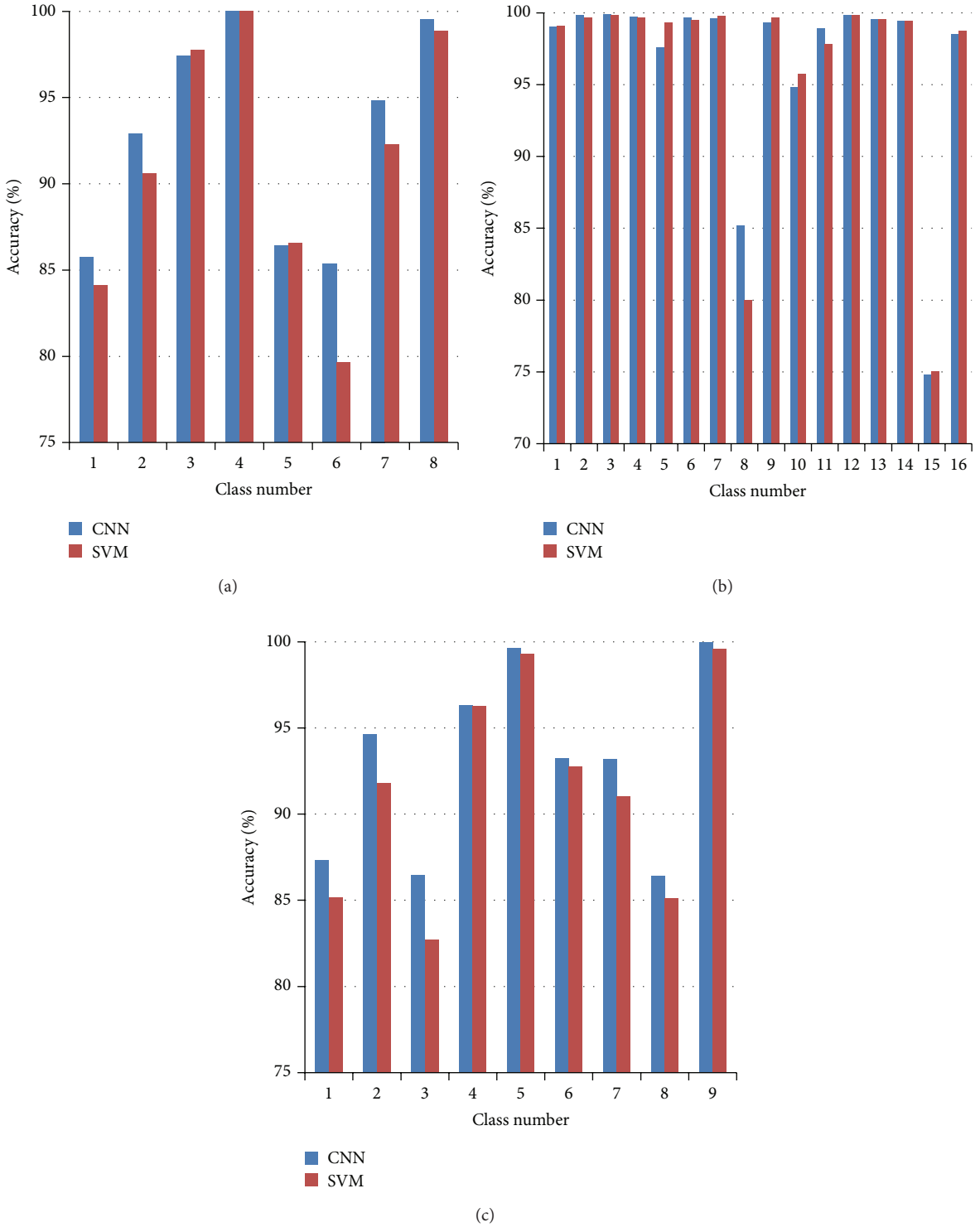

Figure 7: Classification accuracies of all the classes for experimental data sets. From (a) to (c): Indian Pines, Salinas, and University of Pavia. The class number is corresponding to the first column in Tables 1, 2, and 3.

and test time. According to our experiments, it takes only 5 minutes to achieve 90% accuracy on MNIST dataset [32] by using Caffe compared to more than 120 minutes by using our implemented framework.

Figure 9 illustrates the relationship between cost value (see (4)) and the training time for the University of Pavia data set. The value of the loss function is reduced with an increasing number of training iteration, which demonstrates

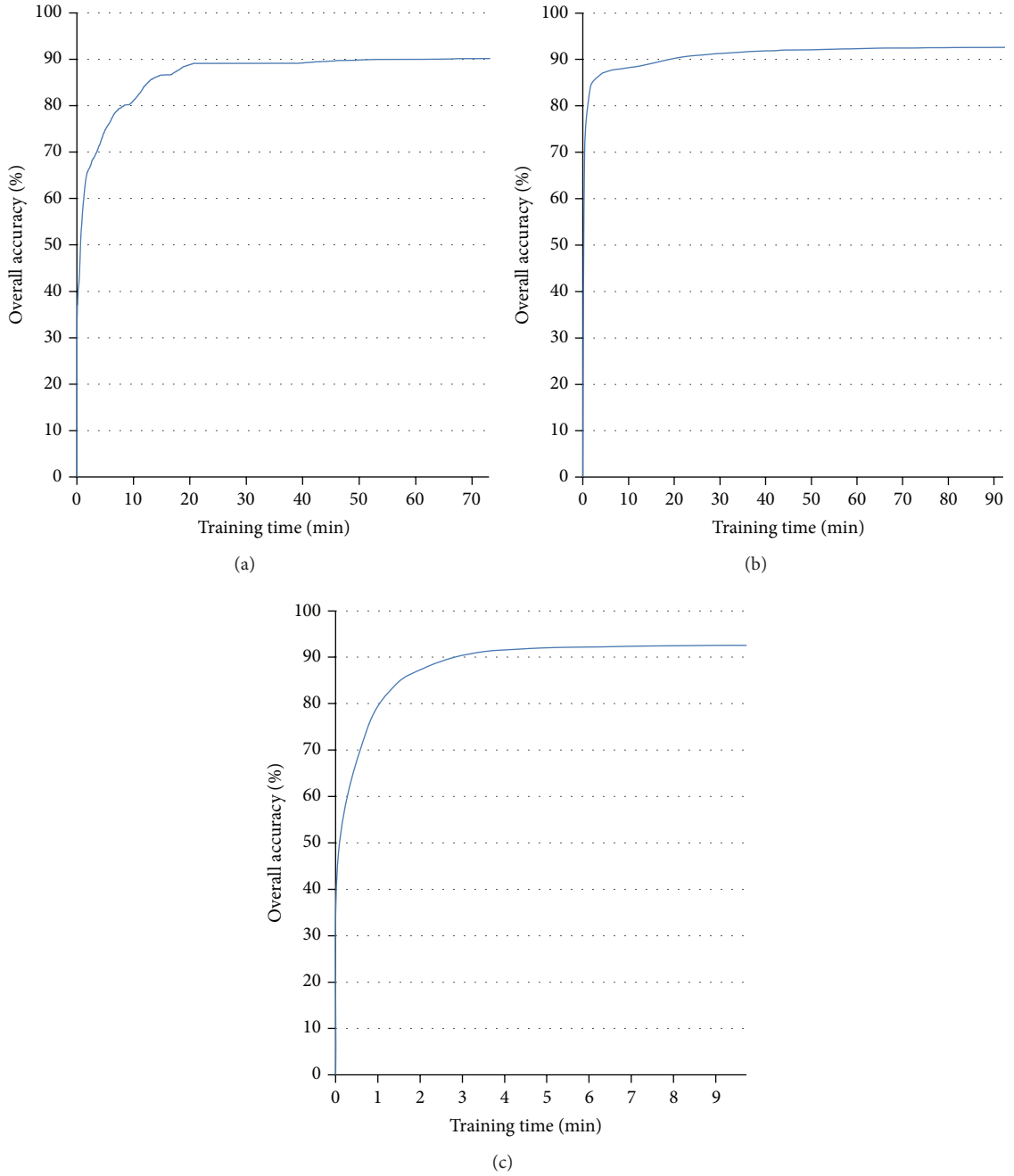

Figure 8: Classification accuracies versus the training time for experimental data sets. From (a) to (c): Indian Pines, Salinas, and University of Pavia. Note that the test time is also included in the training time.

the convergence of our network with only 200 training samples for each class. Moreover, the cost value is still reduced after 5-minute training, but the corresponding test accuracy is relatively stable (see Figure 8(a)), which indicates the overfitting problem in this network.

To further verify that the proposed classifier is suitable for classifying data sets with limited training samples, we also compare our CNN with RBF-SVM under different training sizes on the University of Pavia data set as shown in Figure 10. It is obvious that our proposed CNN consistently provides higher accuracy than SVM. However, although the conventional deep learning-based method [26] can outperform the SVM classifier, it requires plenty of training samples for constructing autoencoders.

To demonstrate the relationship between classification accuracies and the visual differences of curve shapes (see Figure 2), we present the detailed accuracies of our proposed CNN classifiers for the University of Pavia data set in Table 6. In the table, the cell in the ith row, jth column means the percentage of the ith class samples (according to ground

Table 6: The detailed classification accuracies of all the classes for University of Pavia data set.

	Asphalt	Meadows	Gravel	Trees	Sheets	Bare soil	Bitumen	Bricks	Shadows
Asphalt	**87.34%**	0.26%	2.32%	0.00%	0.19%	0.37%	6.25%	3.25%	0.02%
Meadows	0.00%	**94.63%**	0.02%	1.26%	0.00%	4.03%	0.00%	0.06%	0.00%
Gravel	0.53%	0.47%	**86.47%**	0.00%	0.00%	0.00%	0.05%	12.43%	0.05%
Trees	0.00%	2.67%	0.00%	**96.29%**	0.03%	1.01%	0.00%	0.00%	0.00%
Sheets	0.00%	0.09%	0.00%	0.00%	**99.65%**	0.26%	0.00%	0.00%	0.00%
Bare soil	0.12%	6.15%	0.00%	0.10%	0.08%	**93.23%**	0.00%	0.31%	0.00%
Bitumen	6.37%	0.00%	0.35%	0.00%	0.09%	0.00%	**93.19%**	0.00%	0.00%
Bricks	1.90%	0.20%	10.48%	0.00%	0.06%	0.60%	0.34%	**86.42%**	0.00%
Shadows	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	**100.00%**

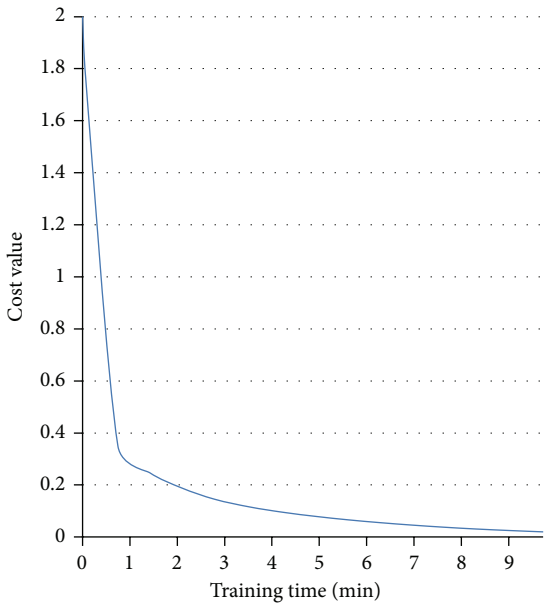

Figure 9: Cost value versus the training time for University of Pavia data sets.

Figure 10: Classification accuracies versus numbers of training samples (each class) for the University of Pavia data sets.

truth) which is classified to the jth class. For example, 87.34% of class Asphalt samples are classified correctly, but 6.25% of class Asphalt samples are wrongly classified to class Bitumen. The percentages on diagonal line are just the classification accuracies of corresponding classes. As for one class, the more unique the corresponding curve shape is, the higher accuracy the proposed CNN classifier can achieve (check the class Shadow and class Sheets in Figure 2 and Table 6). The more similar two curves are, the higher opportunity they are wrongly classified to each other (check the class Gravel and class Bricks in Figure 2 and Table 6). Furthermore, the excellent performance verifies that the proposed CNN classifier has discriminative capability to extract subtle visual features, which is even superior to human vision for classifying complex curve shapes.

Finally, we also implement three other types of neural network architectures for the Indian Pines data set using the same training and test samples. The first one is a simple architecture with only two fully connected layers beside the input layer. The second one is LeNet-5 which is a classic CNN architecture with two convolutional layers. The third one is a conventional deep neural networks (DNNs) with 3 hidden fully connected layers (a 220-60-40-20-8 architecture as suggested in [26]). The classification performance is summarized in Table 5. From Table 5, we can see that our CNN classifier achieves the highest accuracy with competitive training and testing computational cost. LeNet-5 and DNNs cost more time to train models due to their complex architecture, but limited training samples restrict their capabilities of classification (only 20% samples selected for testing in [26] compared with 95% in our experiment). Another reason for the difficulty that deeper CNNs and DNNs face to achieve higher accuracies could be that the HSI lacks the type of high frequency signal commonly seen in the computer vision domain (see Figure 2).

5. Conclusion and Future Work

In this paper, we proposed a novel CNN-based method for HSI classification, inspired by our observation that HSI classification can be implemented via human vision.

Compared with SVM-based classifier and conventional DNN-based classifier, the proposed method could achieve higher accuracy using all the experimental data sets, even with a small number of training samples.

Our work is an exploration of using CNNs for HSI classification and has excellent performance. The architecture of our proposed CNN classifier only contains one convolutional layer and one fully connected layer, due to the small number of training samples. In the future, a network architecture called a Siamese Network [33] might be used, which has been proved to be robust in the situation where the number of training samples per category is small. Some techniques, such as Dropout [34], can also be used to alleviate the overfitting problem caused by limited training samples. Furthermore, recent researches in deep learning have indicated that unsupervised learning can be employed to train CNNs, reducing the requirement of labeled samples significantly. Deep learning, especially deep CNNs, should have great potentiality for HSI classification in the future. Moreover, in the current work, we do not consider the spatial correlation and only concentrate on the spectral signatures. We believe that some spatial-spectral techniques also can be applied to further improve the CNN-based classification. At last, we plan to employ efficient deep CNN frameworks, such as Caffe, to improve our computing performance.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported jointly by the National Natural Science Foundation of China (nos. 61371165 and 61302164), the 973 Program of China (no. 2011CB706900), the Program for New Century Excellent Talents in University under Grant no. NCET-11-0711, and the Interdisciplinary Research Project in Beijing University of Chemical Technology. Wei Hu and Fan Zhang are also supported by the Beijing Higher Education Young Elite Teacher Project under Grant nos. YETP0501 and YETP0500, respectively.

References

[1] D. Landgrebe, "Hyperspectral image data analysis," *IEEE Signal Processing Magazine*, vol. 19, no. 1, pp. 17–28, 2002.

[2] G. M. Foody and A. Mathur, "A relative evaluation of multiclass image classification by support vector machines," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 6, pp. 1335–1343, 2004.

[3] Y. Tarabalka, J. A. Benediktsson, and J. Chanussot, "Spectral-spatial classification of hyperspectral imagery based on partitional clustering techniques," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 8, pp. 2973–2987, 2009.

[4] W. Li, S. Prasad, F. James, and B. Lour, "Locality-preserving dimensionality reduction and classification for hyperspectral image analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 4, pp. 1185–1198, 2012.

[5] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 8, pp. 1778–1790, 2004.

[6] J. A. Gualtieri and S. Chettri, "Support vector machines for classification of hyperspectral data," in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS '00)*, vol. 2, pp. 813–815, IEEE, July 2000.

[7] G. Mountrakis, J. Im, and C. Ogole, "Support vector machines in remote sensing: a review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 3, pp. 247–259, 2011.

[8] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Spectral-spatial classification of hyperspectral data using loopy belief propagation and active learning," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 51, no. 2, pp. 844–856, 2013.

[9] P. M. Atkinson and A. R. L. Tatnall, "Introduction neural networks in remote sensing," *International Journal of Remote Sensing*, vol. 18, no. 4, pp. 699–709, 1997.

[10] L. Bruzzone and D. F. Prieto, "A technique for the selection of kernel-function parameters in RBF neural networks for classification of remote-sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 37, no. 2, pp. 1179–1184, 1999.

[11] F. Ratle, G. Camps-Valls, and J. Weston, "Semisupervised neural networks for efficient hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 5, pp. 2271–2282, 2010.

[12] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[13] K. Fukushima, "Neocognitron: a hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, no. 2, pp. 119–130, 1988.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[15] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI '11)*, vol. 22, pp. 1237–1242, July 2011.

[16] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proceedings of the 7th International Conference on Document Analysis and Recognition*, vol. 2, pp. 958–963, IEEE Computer Society, Edinburgh, UK, August 2003.

[17] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks," in *Proceedings of the International Joint Conference on Neural Network (IJCNN '11)*, pp. 2809–2813, IEEE, August 2011.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Advances in Neural Information Processing Systems 25 (NIPS '12)*, pp. 1097–1105, 2012.

[19] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '12)*, pp. 3642–3649, IEEE, June 2012.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14)*, pp. 580–587, IEEE, June 2014.

[21] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.

[22] P. Sermanet, S. Chintala, and Y. LeCun, "Convolutional neural networks applied to house numbers digit classification," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR '12)*, pp. 3288–3291, IEEE, November 2012.

[23] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: closing the gap to human-level performance in face verification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14)*, pp. 1701–1708, Columbus, Ohio, USA, June 2014.

[24] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Proceedings of the 38th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '13)*, pp. 8614–8618, IEEE, Vancouver, Canada, May 2013.

[25] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '12)*, pp. 4277–4280, IEEE, March 2012.

[26] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu, "Deep learning-based classification of hyperspectral data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 6, pp. 2094–2107, 2014.

[27] I. Sutskever and G. E. Hinton, "Deep, narrow sigmoid belief networks are universal approximators," *Neural Computation*, vol. 20, no. 11, pp. 2629–2636, 2008.

[28] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[29] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*, pp. 9–48, Springer, Berlin, Germany, 2012.

[30] J. Bergstra, F. Bastien, O. Breuleux et al., "Theano: deep learning on GPUs with python," in *Proceedings of the NIPS 2011, Big Learning Workshop*, pp. 712–721, Granada, Spain, December 2011.

[31] Y. Jia, E. Shelhamer, J. Donahue et al., "Caffe: convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678, ACM, Orlando, Fla, USA, November 2014.

[32] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998, http://yann.lecun.com/exdb/mnist/.

[33] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, vol. 1, pp. 539–546, IEEE, June 2005.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.